Higher Technical School of Computer Engineering

Master's in SOFTWARE ENGINEERING AND ARTIFICIAL INTELLIGENCE

# Proactive network load distribution and link utilization optimization using machine-learning techniques

# Distribución proactiva de la carga de red y optimización de la utilización de los enlaces empleando técnicas de aprendizaje computacional

Written by:
Fakhraddin Mohammed

Tutored by:
Ezequiel López Rubio
Department of Computer Languages and Computer Science.
University of Málaga.
Bulevar Louis Pasteur, 35. 29071 Málaga. Spain.

and

Juan Carlos Burguillo Rial
Department of Telematics Engineering.
University of Vigo.
Campus Universitario de Vigo. 36310 Vigo. Spain.

University of Málaga
Málaga, SPAIN (July 2017)

**Abstract.** Flow path prediction is an important capability for environments with multiple paths between geographically separated network branches. In these cases, the main purpose is to efficiently distribute the load on sensitive and critical links. The current work proposes a methodology for the use of three powerful state-of-the-art machine learning techniques: Artificial neural Networks (ANN), Support Vector Machines (SVM) and Decision Trees (DT). This methodology could be used as a network performance analysis framework to handle the optimization of edge links. In order to get the higher accuracy in the results, the project environment has been implemented on a real lab to emulate a real-like situation for measuring and evaluating the predictors. Traffic conditions and flow transmissions have been established using Cisco's IOS XE's built-in features on edge router appliances. Then, performance measurements of the network were gathered using SNMP and NetFlow to input the data collector. Different structures of ML models were trained and validated over collected data, flow transfer statistics and path metrics. The phase followed a 66%–34% training/validation split approach by repeated holdout method. Predictors reported an average accuracy of more than 90% on user data in well-tuned structure sets. A significant improvement has been observed in network performance and utilization of edge links after applying a ML policy. Almost all of the flow pockets routed, using most appropriate outgoing interface, insured better utilization and very less loss rate on critical edge links. Similar improvements in results also experienced in peak time traffic and rush hour situations. In comparison with other passive techniques, the flexibility and accuracy of the current methodology lead to recommend its extension to larger environments for optimization of real-time flow distribution.

**Keywords:** Network Load Distribution, Link Utilization, Machine Learning, IP Flow, Path Prediction, Artificial Neural Networks, Support Vector Machines, Decision Trees

**Resumen.** Poder predecir la mejor conexión de flujo saliente en una red, para la finalidad de distribución de carga en enlaces sensibles, es una capacidad importante para ambientes donde existen múltiples direcciones entre redes separadas. En esta disertación, se propone una metodología para el uso de tres poderosas técnicas de aprendizaje automático: "Artificial Neural Network", "Support Vector Machines" y "Decision Trees", que pueden usarse como análisis en el marco del rendimiento de red. El entorno del proyecto se ha aplicado en una laboratorio real para emular un caso de estudio real para medidas y evaluación de indicadores de cara a obtener una precisión más alta en los resultados. Las condiciones del tráfico y de los flujos se han establecido utilizando características internas de CISCO IOS XE y el rendimiento de medidas de la red mediante SNMP y NetFlow como aportación al colector de datos. Los diferentes modelos de aprendizaje automático han sido validados sobre los datos recolectados, las estadísticas de trasferencia de flujo y métricas de distancias. Se ha utilizado un 66% - 34% entrenamiento/validación por un procedimiento de entrenamiento repetido. Los indicadores resultantes revelaron una precisión de más de un 90% usando datos de usuario en los modelos bien entrenados. Se observa una mejora significativa en el rendimiento de redes y enlaces críticos, después de aplicar la política de aprendizaje automático; para que casi todos los paquetes de flujo dirigidos utilizando interfaces salientes apropiadas, resultando en un mejor uso y una menor porcentaje de pérdida de paquetes. Se observan muchas mejoras similares en los resultados durante horas de mayor tráfico y situaciones de hora punta. Además, la flexibilidad en la precisión de la metodología propuesta, en comparación con otras técnicas pasivas de medición del rendimiento de enlaces críticos, nos llevan a recomendar su aplicación en entornos más grandes para optimizar distribuciones de flujos en tiempo real.

**Palabras clave:** distribución de la carga de la red, uso del acoplamiento, aprendizaje de la máquina, flujo del IP, predicción de la trayectoria, redes neurales artificiales, Máquinas de vectores de soporte, Árboles de decisión

## 1 Introduction

The complexity of computer networks continues to increase as today's IP networks, such as ISPs and global enterprise networks, become larger, and management of these networks demands huge amount of time and efforts. Employment of new technologies, protocols and optimizations techniques at all network layers can improve network throughput, reliability and security, and even able to replace traditional network management systems, which rely on conventional procedures for decision-making. One solution is to manually utilize new techniques such as proactive performance testing by network administrators. However, they can be very time consuming and hence cannot keep up with the emergence of new administrative approaches. Recent research works focus on automatically generating accurate rules using machine-learning techniques (see [1] and [2]). In order to build an automatic administrative approach and proactive performance monitoring system, first we need to define basic network management requirements such as throughput, flow quality, link capacity measurements, etc. Next, we will design a compatible machine learning based method that meets these network requirements in an operational network.

Today's business demands requires companies to expand their networks to different geographically separated areas to be able to handle and carry out a huge mixture of traffic flows from a wide range of diverse applications and customers. To do so most of enterprises have to use leased lines and external ISPs to provide them connectivity. Improving IP connectivity performance through external links and management of various traffic patterns can turn into big challenges. The ability to accurately classify network traffic and map them into the different types of applications flows that each has its own statistic and quality of service priorities, is a main key in proactive link capacity planning, as these core links are like their vital artery and are critical to the operations and management of such networks.

P2P file sharing traffic, on-line gaming, sudden growth in application traffics, DDoS attacks, security threats such as viruses and exploits, and other issues can exhaust the link bandwidth and cause drop and loss in sensitive data. Traffic engineering requires better flow routing and prediction of their requirements to manage the bandwidth consumption. Since the underlay networks of ISPs and lease line providers, are hidden from enterprises, identifying daily emerging traffic quality and end-to-end link performance to forecast and adapt to their application trends and needs, and planning their network capacity accordingly, can also be a big challenge due to use of conventional link monitoring systems.

Capacity planning and throughput optimization for the critical links (e.g., core network, Datacenter, Storage Area Network (SAN) and Wide area Network (WAN) links) is the process of ensuring that sufficient bandwidth is provisioned such that the expected targets of delay, jitter, loss, and availability can be met. The simple and traditional planning processes use passive measurements of link utilization statistics that are analyzed by network engineers, and then configuration profiles are built manually. However, traffic demand changes along time, as it is a dynamic measure, and even issues like scheduled backups, remote storage traffics, virus scanner updates, mails server problems, malware outbreaks and hacking attempts can cause more unexpected spikes in network traffic. Conventional planning may not be enough to ensure that the links are still sufficiently provisioned to meet the best utilization when network element (e.g., links and nodes) failures occur. Even not, sometimes this may result in more capacity being provisioned than actually needed. Accurate network load distribution and link performance optimization in a proactive manner can help service providers to address the requirements of monitoring service level agreements (SLAs) for business customers and other profit-generating uses, as well as traffic policing and policing for performance-sensitive flows.

Networks are inherently distributed systems, where each node (i.e., switch, router) has only a partial view and control over the complete system. Most of link performance measurement methods have been developed for monitoring at the level of end hosts or routing equipment, which do not provide top level view and do not meet the accuracy requirements of end-to-end link management tasks. Even if they do provide overall image of core links of an enterprise, they cannot provide accurate time based data and seasonality statistics. Learning from nodes that can only view and act over a minor part of the system is very complex, particularly if the goal is to exercise the performance of the links that built of top a hidden underlay infrastructure. The simplest example for the processes of core link's usage optimization and efficient routing is the employment of passive measurements of utilization statistics and then applying policies per node/link. Policies can be defined by measuring each link's delay, jitter, loss, and availability statistics to ensuring that quality of service (QoS) targets for specific traffic types are met. However, there is need to have a richer view of the network compared to what is possible with single node or conventional network management approaches. Most of modern data plane devices, such as routers and switches, are shipped with enhanced computing and storage capabilities, which can provide real-time packet and flow information and monitoring data to a centralized analytics machine like SNMP[1] based network monitoring systems (NMS) or Netflow[2] based statistics collectors. In the case of multi-homed ISP connectivity, where the underlay transit network infrastructure is not known to overlay edge devices (Figure 1) - for instance, a company with geographically distributed branches that connects them through the Internet –, these real time data can be used to model the behavior of the underlay network.

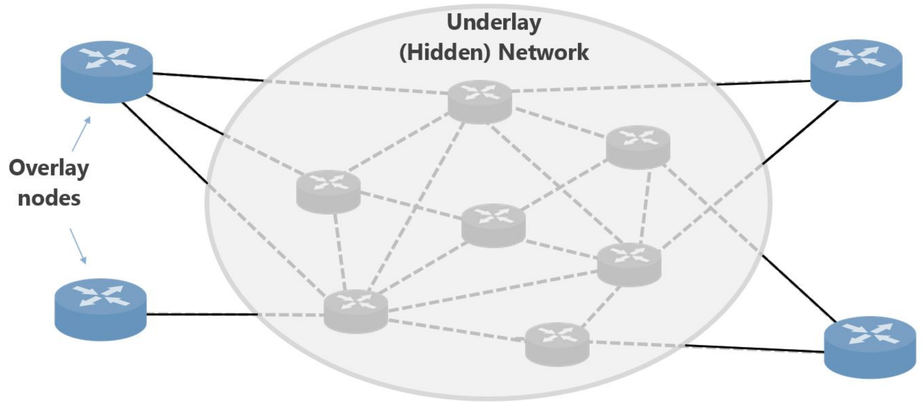


**Fig. 1.** Overlay Network with a hidden Underlay

Even though multiple connections bind enterprise's WAN/edge routers to the underlay transit layer, these edge routing nodes have no control over the underlay forwarding plain. However, they can distribute and balance the traffic over the different connections to other edge routers (located in another geographical areas) using several control plane techniques and protocols (e.g. IGP over GRE Tunneling[3], VXLAN[4], LISP[5].These edge nods are almost the only way

[1] Simple Network Management Protocol
[2] RFC 3954 - Cisco Systems NetFlow Services Export Version 9 at: `https://www.ietf.org/rfc/rfc3954.txt`
[3] RFC 1701 - Generic Routing Encapsulation (GRE) at: `https://tools.ietf.org/html/rfc1701`
[4] RFC 7348 - Virtual eXtensible Local Area Network (VXLAN) at: `https://tools.ietf.org/html/rfc7348`
[5] RFC 6830 - The Locator/ID Separation Protocol at: `https://tools.ietf.org/html/rfc6830`

where the overlay network administrators can control and monitor [of course, to a certain extent] the traffic load distribution and link utilization by defining path forwarding policies over the underlay transit infrastructure, therefore an efficient configuration of edge links is critical. It is difficult for overlay network engineers to build a model to optimize link utilization by improving path performance to force each traffic flow to take a specific route; because the topology and the routing policy of the underlay network are not known. Meanwhile, in the case of large and complex overlay environments, these passively built statistical or theoretical models most probably may not be able to correctly predict the behavior of underlay network.

A machine-learning algorithm could detect seasonality and periodical changes in networks, e.g. see in [3] and [4]. Real-time flow statistics can be collected and analyzed by ML algorithms, to be used for example to classify the traffic flows, dynamically anticipate congestion and take a proactive measures to divert or shrink some of the flows into an alternate path. (see [5], [6], [7] and [8] for ML-based traffic classification techniques).

In this case, Machine learning algorithms can help to model hidden underlay infrastructure by examining the statistical information of flows in a roundtrip fashion over the transit network. This can be achieved by the means of observing how the returning flow behaves in relation to a sending flow (i.e., $f$(routing decisions,data flows)= performance ). Such as for example, if two identical flows be transferred over different edge links, -which are connected to different paths inside underlay network- ML techniques can learn the performance difference per flow per link and determine the batter outgoing interface for that specific flow and therefore recommend traffic forwarding policies to improve flow quality as well as link utilization.

In order to measure, verify and monitor quality of service (QoS) for the traffic over end-to-end IP layer in an overlay network, one method is to gather each flow's statistics like roundtrip duration, delay and latency, jitter and packet loss, etc., per link per direction and send them to a centralized SNMP/NetFlow based collector for further analyze and modeling. These are the basic information of network traffic, and can be collected using UDP/TCP based flow generator probes on edge nodes, which sending data over the underlay network to other edge routers to measure roundtrip statistics.

In this work, based on practical approaches and methods for traffic in IP networks [6] [7] [8] (also see in [9]), flow-level data derived from IP and transport-layer packet headers. Then, using these flow records as ground truth, supervised machine learning (ML) techniques employed to train a model that accurately predicts the probability of choosing the appropriate outgoing interface (edge link) per IP stream. Presented methodology can be applied in a proactive way to continuously gather, train and predict best path per flow over wireline connections using three state-of-the-art machine learning techniques (ANN[9], SVM[10], DT[11]).

Prior works related to the concept and methodology of this work has presented in section 2. Section 3 explains proposed methodology in detail with step-by-step demonstration and implementation of its four blocks including Data Gathering, Preprocessing, Supervised ML and Evolution. Section 4 discuss about observations and experimental results by comparing flow metrics and statistics before and after applying new forwarding rules (policies that generated using prediction results of ML algorithm). Finally, section 5 summarize the contributions of this dissertation and identify possible directions of future work.

[6] Flexible NetFlow Configuration Guide, at: `http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/fnetflow/configuration/xe-3s/fnf-xe-3s-book/fnf-fnetflow.html`

[7] RFC 2720 : N. Brownlee. Traffic Flow Measurement: Meter MIB. IETF, RFC 2720, October 1999

[8] RFC 6374: D. Frost, S. Bryant. Packet Loss and Delay Measurement for MPLS Networks. IETF. RFC 6374

[9] Artificial Neural Network

[10] Support Vector Machine

[11] Decision Trees

## 2 Previous works

In this section, studies related to the problems addressed in this work - proactive load distribution (= efficient routing) and optimizing the utilization of critical links (= improving throughput quality) - are presented. Several measurement studies of the network traffic (e.g. Intranet or Internet) have been conducted to find out how performance measurements of an IP flow such as delay, loss, jitter and throughput differ over time for given routing paths. These studies are highly considerable for predicting transfer metrics (e.g. routing path, forwarding priority . . . ) of an IP stream, as they provide top view of whether any certainty exists in network performance.

Barford and Crovella's application of constructing and profiling critical path analysis for file transfer (a tool called tcpeval that automates critical path analysis), provided a high detailed outlook on how delay, queuing and loss relate to flow performance. They showed that factors like packet propagation, network variation (e.g., queuing at routers or route fluctuation), packet losses, and delays can cause an obvious and measurable change in total transfer latency at both flow sender and receiver[10].

Paxson in [11] studied the behavior of loss, delay, and bottleneck bandwidth on Internet paths by specifying the outbreak of uncommon network events such as out-of-order delivery and packet corruption. Their results showed that packet loss was changing significantly at short intervals, happening typically in periods longer than 200ms, due to drop tail queues at network nodes. They also found that delay was changing between 0.1 and 1 sec, as there were very asymmetric network paths, however, bottleneck bandwidth between a pair of end-hosts was usually stable over time.

Zhang et al. in [12] studied the distribution of rates at which flows transmit data by examining Internet flow rates and the relationship between the rate and other flow characteristics such as size and duration. They found skewness and deviation in flow distribution; however, it was not as significant as the distribution of flow sizes. Results showed that flow rates and flow sizes are interrelated, signifying that the size of data transmitted is not basically absolute, but rather depends on the performance an end-host can get from the network.

Zhang et al. in [13] by exploring three different notions of constancy: mathematical, operational, and predictive they represented highly precise degree of stability for loss, delay, and throughput in network paths. Mathematical constancy, which exists if a set of measurements can be described by a single, time-invariant mathematical model, operational constancy, which determines the granularity of change of a metric needed to have a meaningful impact on application performance, and predictive constancy, which indicates whether time-series analysis of past measurements is useful for predicting future values such as throughput.

Basic mechanisms for implementations of modern network flow measurements, determined by the work of Jacobson and Karels [14], and from then on, it was obvious that also many other elements affect TCP/UDP file transfer performance. Generally, these take in the TCP implementation, the underlying network infrastructure, and the dynamism of the traffic distribution over the links, on the path between senders to receivers. Many research works have modeled TCP throughput using analytical procedures, which consider some of the above-mentioned factors, with later research models trying to advance the accuracy of the earlier ones by using history-based time series data to predict TCP Throughput as well as other flow quality elements.

One of the major works of TCP Throughput via analytical formula has done by Ott et al in [15]. They deduced the static distribution of the TCP congestion window size by approximating the discrete-time process of window size evolution with a continuous-time process. By taking only the congestion avoidance phase in account and assuming that losses are independent per flow, they concluded the well-known inverse square root relationship between throughput and packet loss, to show that throughput is proportional to $\frac{1}{\sqrt{packet\ loss}}$. Mathis et al. in [9] showed an improved form of the model in [16] based on the assumption that loss is intervallic, and indicated that throughput is proportional to $\frac{Maximum\ Segment\ Size}{RTT * \sqrt{packet\ loss}}$.

Previously discussed works and their proposed models, including Altman et al. in [17], suppose that packet streams are available to measure end-to-end per-flow metrics such as packet loss, latency and RTT. Goyal et al. in [18] assumes the situation where the only available statistics is via SNMP information bases, where instead of getting information from individual flows, they gathered aggregated traffic information from routers. They adjust the model discussed in Padhye [19] to derive and measure metrics collected from SNMP MIBs. Arlitt et al. in [20] implemented a heuristic based model for predicting the throughput of a TCP stream based on the transfer size and the RTT of the first packet exchange of the transmission.

In addition to analytical models of flow metrics, several other research works have used time-series approaches for analysis of past flow statistics on a network path to predict future metrics, e.g. throughput.

A significant relationship of throughput to the transfer size showed by Lu et al. in [21], and used this correlation for throughput prediction for arbitrary file sizes. It achieved by establishing one small transfer and one large transfer, and predicting throughput for a given size by outcome of the measurements on the throughput of these two transfers.

An issue with time series approaches is that they produce a huge amount of measurement information of network flows, as in order to forecast the throughput or other flow quality metrics of a transfer of a given size, during a specified period; at some point in the past an equal transfer has to be established and measured. Most history-based methods implement some sort of averaging, so usually the flow transmission should be established and repeated for a period of time. Given changes in path conditions over time, recent historical statistics are more likely to result accurate predictions than older ones. To reach higher accuracy, flows should transferred frequently and as close to prediction time as possible. For the Network Weather Service [22], Swany et al. handle the competing demands of accuracy and low overhead by employing a mixture of large, heavyweight transfers, established infrequently, and small, lightweight transfers, established frequently.

A lightweight machine-learning based approach has introduced by Mirza, et al. in [23] for TCP throughput prediction. They have used history-based transfer statistics and measurements of common path metrics for this prediction.They implemented the prediction model (based on SVR[12]) in a tool called PathPerf[13] and showed highly accurate predictions over various network paths.

[12] Support Vector Regression

[13] `http://wail.cs.wisc.edu/waildownload.py`

## 3 Proposed approach

The overall structure of the methodology and workflow used in this work comprised of four major steps, as depicted in Figure 2.

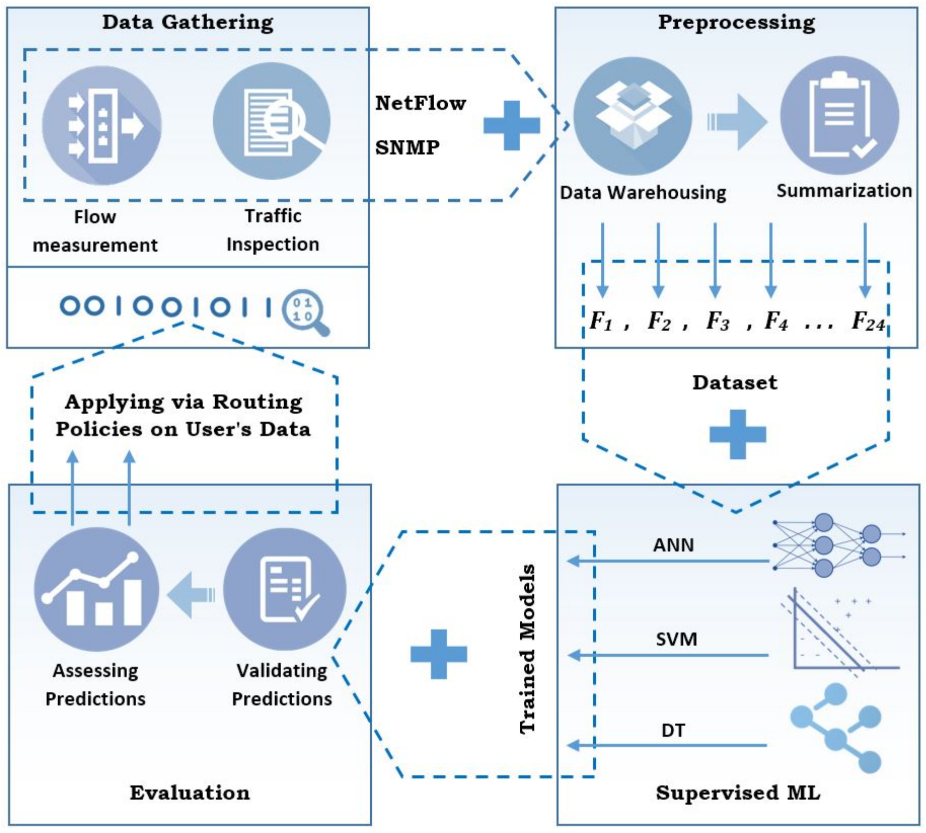


**Fig. 2.** Proposed Methodology

The first step called data gathering related to processing and measurement of real-time traffic to get flow information and roundtrip statistics. Preprocessing is the next step of this workflow where collected data are organized and summarized based on influential variables. Then summarized datasets of all concurrent flows used in Supervised ML step to fed and train three different ML models to help to define efficient routing policy for core links. Finally, in the evolution step, the accuracy of predictions are calculated to apply for implementing policies on user's data.

## 3.1 Data gathering

In order to perform flow optimization on critical links, unlike most of related works where prediction accuracy estimated using simulating software (e.g. NS2, Omnet++, ...) , in this work real routing appliances on a virtual environment tested. Computing resources used for implementation of this system were based on a bare metal SuperMicro server located in the LAB of Information Technology Group (GTI) of university of Vigo, with the specifications listed in Table 1:

**Table 1.** Computing resources

| | | |
|---|---|---|
| **Model** | X9DR3-F | |
| **CPU** | Processor type | Intel® Xeon® CPU E5-2620 v2 @ 2.10GHz |
| | Sockets | 2 |
| | Cores per socket | 6 |
| **Memory** | 64 GB | |
| **Storage** | 3 TB | |

To access to the validity approach of this experiment, overlay and underlay networks deployed using Cisco IOS XE based "CSR 1000v" [14] routers inside VMware ESXi 6.5 hypervisor installed on that server. Figure 3 shows an overview of this implementation.

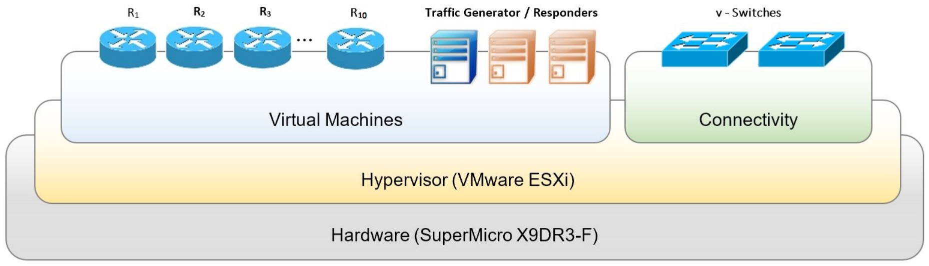


**Fig. 3.** Visualization infrastructure

Overlay network consisted of three office locations, called "Headquarter (HQ) office", "Datacenter (DC) facility" and "Branch Office", each located on different segment to illustrate geographical distance. These locations connected over

[14] `http://www.cisco.com/c/en/us/products/routers/cloud-services-router-1000v-series/index.html`

underlay network via edge routers and links, HQ with one internal router and one edge router with three redundant links (2×10 Mbps and 1×5Mbps) and two other locations each with a edge router connected using a 10 Mbps link. From overlay perspective only its own routers that send and receive traffic are seen, while the underlay network is hidden. For purpose of simulating real operational traffic over network, an open-source traffic generator called "packETH" [15] installed in three locations. In addition, to emulate an Internet like environment, another six routers used to build underlay network with IPv4 addressing scheme and BGP[16] routing protocol, as demonstrated in Figure 4:

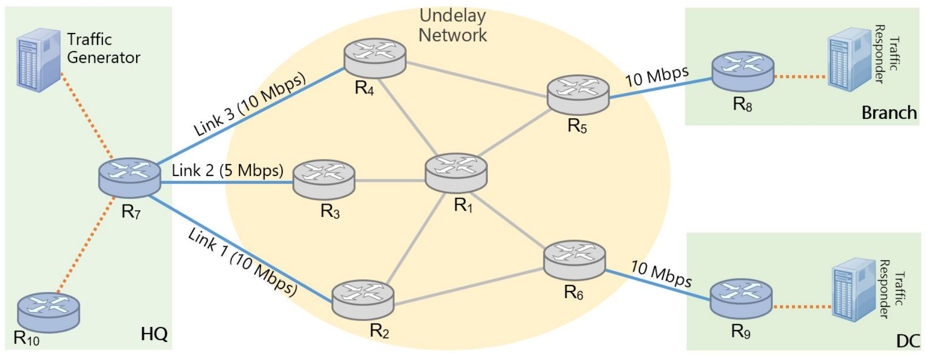


**Fig. 4.** Network infrastructure

Connectivity between routers established by VMware's virtual networking feature, by providing 1 Gbps Ethernet connections. To set the bandwidth of links to the limits mentioned on Figure 4, Traffic policing have configured on underlay network nodes. Meantime to avoid traffic drop due to policing at underlay side, edge routers (R7, R8 and R9) configured to perform traffic shaping to put the flows in queue and send them approximately at same rate of link's bandwidth. These QoS mechanisms illustrated in figure 5 (source: cisco.com). Table 2 shows the result of this configuration, obtained using iPerf[17] application.

To obtain high quality and detailed ground truth data of flow level in each examined direction, the next step of proposed solution, uses a built-in IP SLA

[15] `http://packeth.sourceforge.net/packeth/Home.html`
[16] Border Gateway Protocol
[17] `https://iperf.fr`

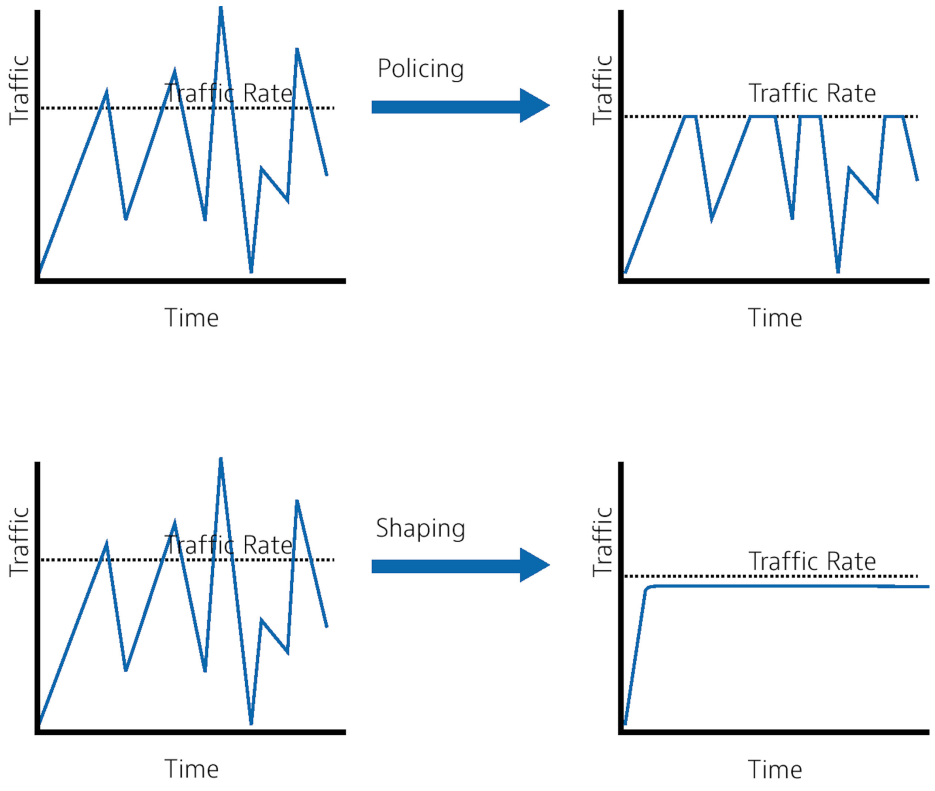


**Fig. 5.** QoS Policing and Shaping techniques

**Table 2.** Link Speed tested by iPerf application

| **Connection Speed** | |
|---|---|
| Link 1 | [ ID] Interval Transfer Bandwidth Retr<br>[ 4] 0.00-10.00 sec 12.4 MBytes 10.4 Mbits/sec 10 sender<br>[ 4] 0.00-10.00 sec 11.6 MBytes 9.72 Mbits/sec receiver |
| Link 2 | [ ID],Interval,Transfer,Bandwidth,Retr<br>[ 4] 0.00-10.00 sec 6.28 MBytes 5.27 Mbits/sec 7 sender<br>[ 4] 0.00-10.00 sec 5.82 MBytes 4.88 Mbits/sec receiver |
| Link 3 | [ ID],Interval,Transfer,Bandwidth,Retr<br>[ 4] 0.00-10.00 sec 12.4 MBytes 10.4 Mbits/sec 10 sender<br>[ 4] 0.00-10.00 sec 11.6 MBytes 9.72 Mbits/sec receiver |

feature of Cisco IOS XE, called "UDP Jitter Operations" [18] to implement flow sender and responders (or probes) on router nodes.

The data and statistics gathered via these operations then transferred to Network Management System (NMS, sometimes called Network Monitoring System) via SNMP and NetFlow protocols for further classification and analyze. Cisco's UDP jitter operation feature is to analyze round-trip delay, one-way (per direction) delay, one-way jitter, one-way packet loss, and some other connectivity measures in networks that carry traffic in IPv4 or IPv6 packets and diagnoses network suitability for real-time traffic applications such as Critical Business Data, VoIP, Video over IP, or real-time conferencing. Other vendors have similar implementations on their commercial products with different names. Figure 6 shows an overview of the measurement scheme in this work.

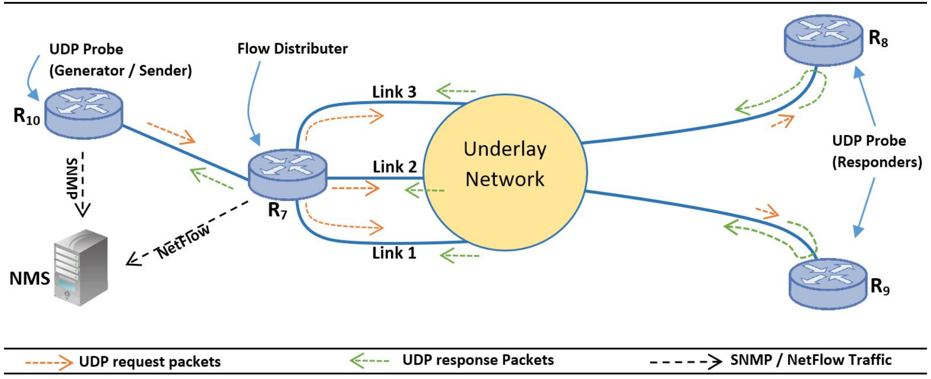


**Fig. 6.** UDP flow patterns

According to Cisco's website [18], Jitter means inter-packet delay variance. When multiple packets are sent consecutively from a source to a destination, for example, 10 msec apart, and if the network is behaving ideally, the destination should receive the packets 10 msec apart. But if there are delays in the network (like queuing, arriving through alternate routes in underlay network, and so on) the arrival delay between packets might be greater than or less than 10 msec. Using this example, a positive jitter value indicates that packets arrived greater than 10 msec apart. If packets arrive 12 msec apart, then positive jitter is 2

[18] Cisco Systems: Configuring IP SLAs UDP Jitter Operations, `http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipsla/configuration/xe-3s/sla-xe-3s-book/sla_udp_jitter.pdf`, August 2016

msec; if packets arrive 8 msec apart, negative jitter is 2 msec. For delay-sensitive networks like VoIP, positive jitter values are undesirable, and a jitter value of 0 is ideal".

Additionally, these UDP operations does more than just monitor jitter. As other measuring data, which replied by a responder are also included, it can be used as a multipurpose data gathering operation. The packets that IP SLAs generate transmit packet-sending and receiving sequence information, and sending and receiving time stamps from the source and the operational target. Based on this information, UDP jitter operations are capable of measuring the following (not limited to):

– Per-direction jitter (source to destination and destination to source)
– Per-direction packet loss
– Per-direction delay (one-way delay)
– Round-trip delay (average round-trip time)

As paths for sending and receiving data is different (asymmetric) in this case, the per-direction data helps to more easily recognize where congestion or other problems are happening in the network. The operation works by initiating artificial (simulated) UDP traffic. Asymmetric probes support custom-defined packet sizes per direction with which different packet sizes can be sent in request packets (from the source node to the destination node) and in response packets (from the destination node to the source node).

The this experiment, the SLA operation sends $N$ number of UDP packets, each of size $S$, $T$ milliseconds apart, from the sender router node ( R10) to other responder routers (R8 and R9), at a specified frequency of $F$. In response, UDP packets of size P is sent back from responder routers R8 and R9 to the R10, via underlay network paths. 24 UDP operation flows implemented to accurately emulate the IP service that represents real situation of data traffic of an enterprise, (see flow IDs in Table 3).

As the packet types all are UDP based, in order to separate different application types and apply quality of service (QoS) techniques, Type of Service (ToS) field of each packet set to appropriate value while initiating it from sender. On edge routers (R7, R8 and R9), differentiated services QoS techniques[19] implemented to queue and prioritize flows based von their ToS value. Higher the ToS

[19] RFC-2475 : D. Frost, S. Bryant. Packet Loss and Delay Measurement for MPLS Networks. IETF. RFC 2475, S. Blake and et all. An Architecture for Differentiated Services

**Table 3.** UDP Flows

| **Destination** | **Output link on R7** | **Business Data** | **Network Services** | **Media Streaming / Video** | **VoIP** |
|---|---|---|---|---|---|
| | Link 1 | 10830 | 10840 | 10850 | 10860 |
| Branch | Link 2 | 21830 | 21840 | 21850 | 21860 |
| | Link 3 | 32830 | 32840 | 32850 | 32860 |
| DC | Link 1 | 10930 | 10940 | 10950 | 10960 |
| | Link 2 | 21930 | 21940 | 21950 | 21960 |
| | Link 3 | 32930 | 32940 | 32950 | 32960 |
| ToS Value: | | 96 | 128 | 160 | 192 |
| DSCP Class: | | cs3 | cs4 | cs5 | cs6 |

value has higher priority. Each flow operation consisted of ten packet frames ($N$), each packet with a payload size of 1480 bytes ($S$) (total payload size of each UDP operation flow = 15000 bytes) and generated every 10 msec ($T$), with the operation is repeated every 2 seconds ($F$). The timeout for each operation (10 packets) was 2000 msec. (see table 4).

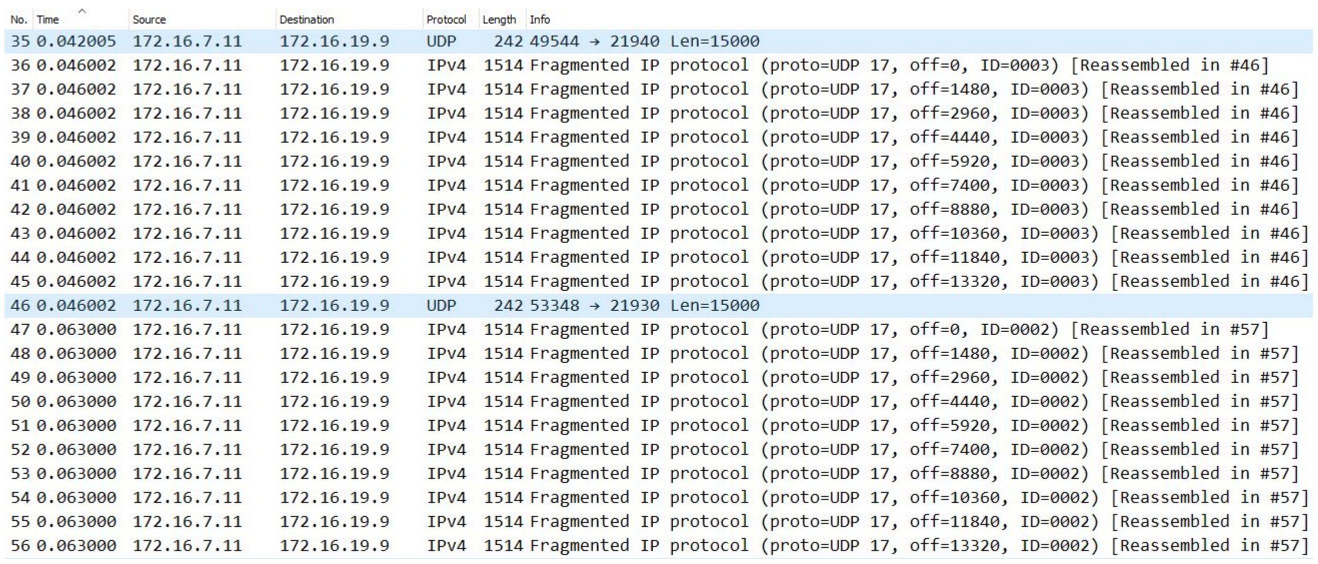
**Table 4.** Flow Details

| **Parameter** | **Value** |
|---|---|
| Number of packets ($N$) | 10 packets |
| Payload size per request packet ($S$) | 1480 bytes |
| Payload size per response packet ($P$) | 1480 bytes |
| Time between packets, in milliseconds ($T$) | 10 msec |
| Elapsed time before the operation repeats, in seconds ($F$) | 2 seconds |
| Timeout of each flow | 2000 msec |

Each IP SLAs operation (for example, IP SLAs operation ID 10830) repeats at 2 sec frequency for the lifetime of the operation. Figure 7 shows two consecutive flows (21930 and 21940) captured on the incoming direction to R7 of the link between R10 to R7, captured by Wireshark [20]. Also, see figure 8 for a detailed view of one the packets of flow ID 21940.

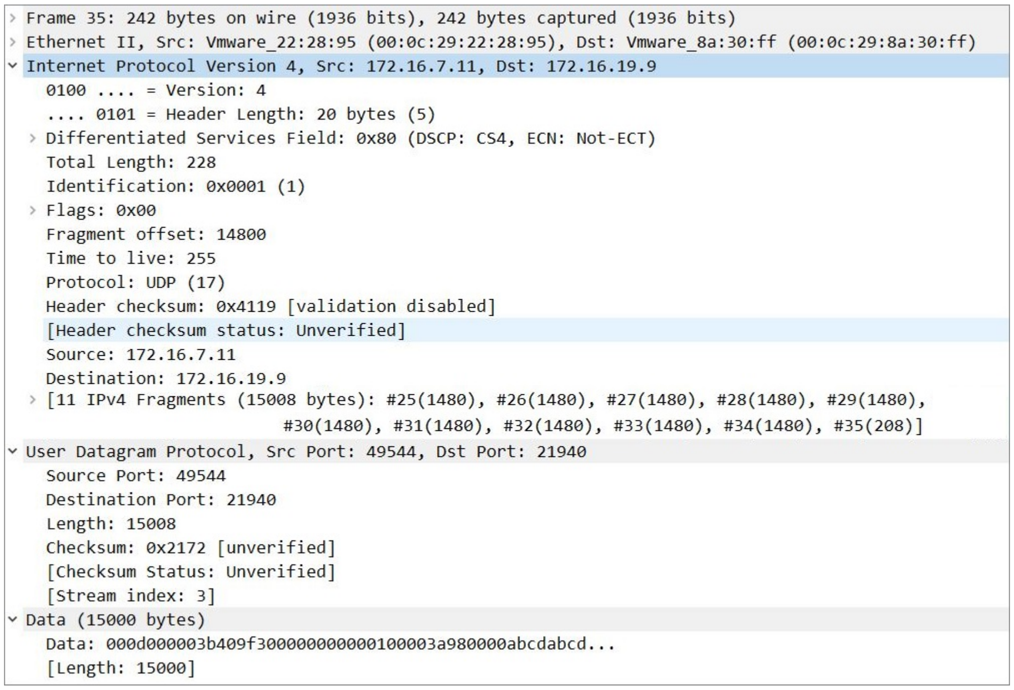
According to topology shown previously in Figure 4, routing paths inside underlay network per flow for out and in directions listed in table 5. Each router

[20] `https://www.wireshark.org/`



```
No. Time       Source       Destination  Protocol Length Info
35 0.042005  172.16.7.11  172.16.19.9  UDP      242 49544 → 21940 Len=15000
36 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=0003) [Reassembled in #46]
37 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=0003) [Reassembled in #46]
38 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=0003) [Reassembled in #46]
39 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=0003) [Reassembled in #46]
40 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=5920, ID=0003) [Reassembled in #46]
41 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=7400, ID=0003) [Reassembled in #46]
42 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=8880, ID=0003) [Reassembled in #46]
43 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=10360, ID=0003) [Reassembled in #46]
44 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=11840, ID=0003) [Reassembled in #46]
45 0.046002  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=13320, ID=0003) [Reassembled in #46]
46 0.046002  172.16.7.11  172.16.19.9  UDP      242 53348 → 21930 Len=15000
47 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=0002) [Reassembled in #57]
48 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=1480, ID=0002) [Reassembled in #57]
49 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=2960, ID=0002) [Reassembled in #57]
50 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=4440, ID=0002) [Reassembled in #57]
51 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=5920, ID=0002) [Reassembled in #57]
52 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=7400, ID=0002) [Reassembled in #57]
53 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=8880, ID=0002) [Reassembled in #57]
54 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=10360, ID=0002) [Reassembled in #57]
55 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=11840, ID=0002) [Reassembled in #57]
56 0.063000  172.16.7.11  172.16.19.9  IPv4    1514 Fragmented IP protocol (proto=UDP 17, off=13320, ID=0002) [Reassembled in #57]
```

**Fig. 7.** Packet capture of two sample flows



```
> Frame 35: 242 bytes on wire (1936 bits), 242 bytes captured (1936 bits)
> Ethernet II, Src: Vmware_22:28:95 (00:0c:29:22:28:95), Dst: Vmware_8a:30:ff (00:0c:29:8a:30:ff)
v Internet Protocol Version 4, Src: 172.16.7.11, Dst: 172.16.19.9
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x80 (DSCP: CS4, ECN: Not-ECT)
    Total Length: 228
    Identification: 0x0001 (1)
  > Flags: 0x00
    Fragment offset: 14800
    Time to live: 255
    Protocol: UDP (17)
    Header checksum: 0x4119 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.7.11
    Destination: 172.16.19.9
  > [11 IPv4 Fragments (15008 bytes): #25(1480), #26(1480), #27(1480), #28(1480), #29(1480),
                        #30(1480), #31(1480), #32(1480), #33(1480), #34(1480), #35(208)]
v User Datagram Protocol, Src Port: 49544, Dst Port: 21940
    Source Port: 49544
    Destination Port: 21940
    Length: 15008
    Checksum: 0x2172 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 3]
v Data (15000 bytes)
    Data: 000d000003b409f300000000000100003a980000abcdabcd...
    [Length: 15000]
```

**Fig. 8.** Detailed packet capture

inside underlay network configured to be belonged to a separate Autonomous system (AS). AS-Path attribute is the default metric for choosing shortest path over Internet. In this implementation, it makes sure flows being routed over same path in both directions.

**Table 5.** Flow Routing Paths

| **Via link (on R7)** | **Destination** | **Flow ID** | **Request direction** | **Respond direction** |
|---|---|---|---|---|
| Link 1 | Branch | 10830 | R10-R7-R2-R1-R5-R8 | R8-R5-R1-R2-R7-R10 |
| | | 10840 | | |
| | | 10850 | | |
| | | 10860 | | |
| | DC | 10930 | R10-R7-R2-R6-R9 | R9-R6-R2-R7-R11 |
| | | 10940 | | |
| | | 10950 | | |
| | | 10960 | | |
| Link 2 | Branch | 21830 | R10-R7-R3-R1-R5-R8 | R8-R5-R1-R3-R7-R10 |
| | | 21840 | | |
| | | 21850 | | |
| | | 21860 | | |
| | DC | 21930 | R10-R7-R3-R1-R6-R9 | R9-R6-R1-R3-R7-R10 |
| | | 21940 | | |
| | | 21950 | | |
| | | 21960 | | |
| Link 3 | Branch | 32830 | R10-R7-R4-R5-R8 | R8-R5-R4-R7-R10 |
| | | 32840 | | |
| | | 32850 | | |
| | | 32860 | | |
| | DC | 32930 | R10-R7-R4-R1-R6-R9 | R9-R6-R1-R4-R7-R10 |
| | | 32940 | | |
| | | 32950 | | |
| | | 32960 | | |

As seen from table 5, each group of four flows are routed by different routing direction based on their destination address (due to BGP's default path selection algorithm), where some have shorter path therefore have less delay, jitter and loss in both directions. Meanwhile, because of QoS queuing and prioritization implemented on edge routers, extra delay are added to those flows that are less critical. Clock synchronization between sender and receiver probes were very critical to accurately measure time-based information. For this purpose, a time-server implemented along with routers to allow them to communicate and synchronize with each other by Network Time Protocol (NTP).

## 3.2 Preprocessing

Large volume of data involved with these testing operations, as in addition to detailed delay and jitter information, they have the ability to detect in which direction a packet was lost in. they are also calculate statistics about the periods of packet loss. Therefore, an external database used to store gathered information and statistics, including UDP operation results via SNMP protocol from R10 and flow level data on real-time network traffic (e.g. source and destination IP addresses and port numbers, outgoing interface, QoS information ...) by NetFlow protocol from R7. Then these statistics and measures extracted into CSV type datasets to feed Machine-learning algorithm. Cisco's NetFlow has become a de-facto standard for flow based data protocols commonly used in enterprise networks. The derived information by Netflow can be warehoused for later retrieval and analysis in support of proactive network management. Information gathered by Netflow in this experience have indicated in Table 6. Also, an example of NetFlow records shown in Table 7.

**Table 6.** NetFlow features

| **Field** | **Definition** |
|---|---|
| IP ToS | Value in the type of service (ToS) field. |
| Protocol | Protocols type of the packet |
| IP Source Address | Sender IP address of the flow. |
| IP Destination Address | Receiver IP address of the flow. |
| Source Port | Value of the transport layer source port field. |
| Destination Port | Value of the transport layer destination port field. |
| Interface Input | Interface on which the traffic received. |
| Interface Output | Interface on which the traffic transmitted. |
| IP Next Hop Address | IP address of the next hop (or router). |
| Counter Bytes | Number of bytes seen in the flow. |
| Counter Packets | Number of packets seen in the flow. |
| Time Stamp System Uptime First | Time, in milliseconds (when the first packet was switched) |
| Time Stamp System Uptime Last | Time, in milliseconds (when the last packet was switched) |

**Table 7.** Sample NetFlow Records

| **No.** | **Time Stamp** | **Src. IP** | **Dest. IP** | **Protocol** | **Src Port** | **Dest. Port** | **ToS** | **...** |
|---|---|---|---|---|---|---|---|---|
| 11 | 42888.72222 | 172.16.7.11 | 172.16.28.8 | IPv4 | n/a | n/a | 192 | ... |
| 12 | 42888.72292 | 172.16.7.11 | 172.16.19.9 | UDP | 53348 | 21050 | 160 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

Each UDP flow's quality measures resulted by SLA operations, primarily had 24 features, which collected and stored in database via SNMP every 60 seconds for the duration of  68 hours. Operations generated 24 different datasets each with 24 variable and 4060 records (= rows), with the final combined dataset of 97440 (24 x 4060) rows. The dataset then statistically summarized (using R software environment) into 9744 records by the placing the mean of each 10 records (= 10 minutes) into one row, with setting the values of empty records (timeout packets) to the upper limits. This helps to ensure that dataset is not containing errors. Next, dataset limited to the most important variables with those that have practical influence on the traffic, including:

- Flow ID
- Destination
- Outgoing Interface / edge link
- Timestamp
- Avg. Round Trip Time (RTT)
- Avg. Jitter (Round Trip)
- Avg. Latency Source - Destination
- Avg. Latency Destination – Source
- Avg. Jitter Source – Destination
- Avg. Jitter Destination - Source
- Packets Lost (Percent)

For instance, Table 8 shows two sample rows of dataset after summarization:

**Table 8.** Sample Dataset

| **Flow id (factor)** | **Dest. (factor)** | **Outgoing interface (factor)** | **Avg RTT (msec)** | **Avg Jitter (msec)** | **Packets out of Seq (#)** | **Avg Latency Src_Dest (msec)** | **Avg Latency Dest_Src (msec)** | **Loss rate (%)** |
|---|---|---|---|---|---|---|---|---|
| 10830 | Branch | link_1 | 130.68 | 19.08 | 0.24 | 153.6 | 11.96 | 26.7636 |
| 10950 | Datacenter | link_1 | 4.3 | 1.2 | 0 | 2.1 | 0.4 | 7.5 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

Then to get the quality of each flow to be compared with the lower limits of application usages (VoIP quality, Video Streaming resolution, Biz Data transfer throughput, etc.), two common quality characteristics of IP traffic flows, calculated using below method and then added to the dataset:

1. **TCP throughput:** (in bits per seconds, based on the Mathis et.al. Formula [9], using "Avg-RTT" (RTT), "Max segment size" (MSS) and percentage of "packets lost", as below:
   *Rate ≤ ( MSS/RTT)*(1 / sqrt(p) ), where p is the probability of packet loss:*

```
for (i in 1:nrow(df)){
    df$TCP_throughput[i] = ((1460*8)/(df$Avg_RTT[i]/1000)) *
        (1/(sqrt((df$Packets_Lost[i] * 1480)/15000)))
}
```

   As the segment size of a TCP datagram is a measure of bytes carried out as data, in this experiment, it is equivalent to the size of transport layer's data payload of each packet, which concluded as below (Additionally, as the value of Packets Lost in primary dataset were based on whole flow's payload (15000), it was taken in account in the formula):
   *15000 ÷ 10 (packets per flow) = 1500, then – 20 (IP header) and – 20 (TCP header) = 1460.*

2. **Mean opinion score (MOS):** (based on ITU-T recommendations P.800[21] and P.910[22]), is a measure to get quality of an IP service such as VoIP in networks, calculated by following steps:
   (a) Calculating "Effective Latency"
   *effective-latency = latency + (2 × jitter) + 10.0*

[21] https://www.itu.int/rec/T-REC-P.800-199608-I/en
[22] https://www.itu.int/rec/T-REC-P.910-200804-I/en

```
for (i in 1:nrow(df)){
  df$effective_latency[i] = (2 * df$Avg_Jitter[i])     +
    (max(df$Avg_Latency_Src_Dest[i] , df$Avg_Latency_
    Dest_Src[i])) + 10
}
```

(b) Calculating "R value" (individual ratings for each flow):
For $effective-latency < 160.0ms : R = 93.2-(effective-latency)/40.0$
For $effective - latency \geq 160.0ms : R = 93.2 - (effective - latency - 120.0)/10.0$
then : $R = R - (2.5 * packet - loss)$:

```
for (i in 1:nrow(df)){

  if( df$effective_latency[i] < 160.0){df$R_value[i] =
    93.2 - ((df$effective_latency[i])/40.0)

  } else if (df$effective_latency[i] >= 160.0){df$R_
    value[i] = 93.2 - ((df$effective_latency[i] - 120.0)
    /10.0)
  }
  df$R_value[i] = df$R_value[i] - (2.5 * df$Packets_Lost
    [i])
}
```

(c) Finally, the MOS score is calculated with the following formula:
For $R \leq 0 : MOS = 1.0$
For $0 < R < 100.0 : MOS = 1 + (0.035 * R) + (0.000007 * R * (R - 60) * (100 - R))$
For $R \geq 100.0 : MOS = 4.5$

```
for (i in 1:nrow(df)){
  if( df$R_value[i] < 0.0){
    df$MOS_score[i] = 1
  } else if (df$R_value[i] < 100.0){
    df$MOS_score[i] = 1 + (0.035 * df$R_value[i]) +
    (0.000007 * df$R_value[i] *(df$R_value[i] - 60 ) *
    (100 - df$R_value[i]))
  } else {
    df$MOS_score[i] = 4.5
  }
}
```

VoIP calls often are in the 3.5 to 4.2 MOS range. Based on user experiences on quality of conversations over networks, MOS scale (or level) defines as below:

| Rating | Label |
|---|---|
| 5 | Excellent |
| 4 | Good |
| 3 | Fair |
| 2 | Poor |
| 1 | Bad |

After above calculation steps, "TCP throughput" and "MOS Score" added to dataset to be used in ML algorithm for training Models. An example of final version of the dataset shown in Table 9:

**Table 9.** A sample of final Dataset

| **Time Stamp** | **Flow id** | **Destination** | **Outgoing interface** | **TCP throughput (#bps)** | **Effective latency (#msec)** | **R value (#)** | **MOS score (#)** |
|---|---|---|---|---|---|---|---|
| 42888.72222 | 10830 | Branch | link_1 | 55001.71 | 201.76 | 18.115 | 1.1991151 |
| 42888.72292 | 10950 | Datacenter | link_1 | 3157610.28 | 14.5 | 74.0875 | 3.782378 |
| ... | ... | ... | ... | ... | ... | ... | ... |

## 3.3 Supervised ML

To address the challenges of obtaining high quality ground-truth data from dataset (Flow level data and measurements), the proposed solution of this work, utilized supervised machine learning to train different models with different structures, to predict appropriate outgoing direction per flow for better link utilization and efficient routing. Models are trained by training/validation split method and their accuracy evaluated via different structures and repeated Hold-Out technique using separate testing dataset. Additionally the whole process for each algorithm repeated several times. ML algorithms written in R software environment:

– Artificial Neural Networks (ANN), using **nnet** package
– Support Vector Machines (SVM), using **e1071** package

– Decision Tress (DT), using **rpart** package

ANN Model trained by a Neural Net algorithm consisted of one hidden layer with various structures. For example, Code listing **"ANN Algorithm"** shows the Neural Net algorithm used of training and validating nine different models (three different values for "neuron size" and three different values for "maximum iteration" attribute).

**Code Listing: ANN Algorithm**

```
# Initiating an empty vector to hold calculated result of 30 iteration:
ANN_ACC_Values <- NULL

# Initiating a matrix to hold Mean and Max accuracies of 9 different ANN models:
ANN_ACC_Matrix = matrix(data=NA, nrow=9, ncol=4)
count = 1

#a "for" loop to define 3 different values of neuron Size:
for (neuron_size in c(50, 75, 100)){

  #a "for" loop to define 3 different values of maxit:
  for (max_itr in c(1100, 1300, 1500)){

     #a "for" loop to perform "HoldOut Method" - 30 repetition :
     for (i in 1:30){

       #Dividing data into Training and Testing sets by random selection of rows from original dataset
       #(2/3 will be place into Training_Set and rest of them into Testing_Set)
        Training_rows <- sample(1:nrow(df), (2/3)*nrow(df) , replace=F)
        Testing_rows <- setdiff(1:nrow(df), Training_rows)
        Training_set <- df[Training_rows,]
        Testing_set <- df[Testing_rows,]

       # Building an ANN model based on training dataset (dependent variable: "outgoing_interface"):
```

```
      ANN_Model <- nnet(outgoing_interface ~ destination + TCP_throughput + MOS_score , data=Training_set , size=neuron_size , maxit=max_itr , decay=5e-3, trace=FALSE, MaxNWts = "unlimited")

      #Calculating predictions based on the Testing dataset:
      ANN_Prediction <- predict(ANN_Model, Testing_set , type="class")

      #Comparing prediction against testing flow statictics to find out number of matching:
      ANN_matched = 0
      for ( m in 1:nrow(Testing_set)){
          if (Testing_set$outgoing_interface[m] == ANN_Prediction[m]){
              ANN_matched = ANN_matched + 1
          }
      }

      #Calculating the Accuracy rate of each calculation and Storing them in Accuracy vector:
      ANN_Accuracy <- (ANN_matched * 100 ) / nrow(Testing_set)
      ANN_ACC_Values <- append( ANN_ACC_Values , ANN_Accuracy)

    }
    #Stroing Accuracy results of each of 9 different ANN structures in the matrix
    ANN_ACC_Matrix[count,1] <- neuron_size
    ANN_ACC_Matrix[count,2] <- max_itr
    ANN_ACC_Matrix[count,3] <- mean(ANN_ACC_Values)
    ANN_ACC_Matrix[count,4] <- max(ANN_ACC_Values)

    count <- count + 1
  }
}
```

To train the SVM Models, the algorithm used different configuration sets. Code listing **"SVM Algorithm"** shows an example of SVM algorithm used of training and validating the nine different models (three different values for "Cost" and three different values for "Gamma" attributes)

**Code listing: SVM Algorithm**

```r
# Initiating an empty vector to hold calculated result of 30
    iteration:
SVM_ACC_Values <- NULL

# Initiating a matrix to hold Mean and Max accuracies of 9
    different SVM models:
SVM_ACC_Matrix = matrix(data=NA, nrow=9, ncol=4)

count = 1

#a "for" loop to define 3 different values of cost:
for (cost_value in c(2000, 5000 , 10000)){

  #a "for" loop to define 3 different values of gamma:
  for (gamma_value in c(20, 50 , 100)){

    #a "for" loop to perform "HoldOut Method" - 30 repetition:
    for (i in 1:30){

      #Dividing data into Training and Testing sets by random
    selection of rows from original dataset
      #(2/3 will be place into Training_Set and rest of them
    into Testing_Set)
      Training_rows <- sample(1:nrow(df), (2/3)*nrow(df) ,
    replace=F)
      Testing_rows <- setdiff(1:nrow(df), Training_rows)
      Training_set <- df[Training_rows,]
      Testing_set <- df[Testing_rows,]

      # Building an SVM model based on training dataset (
    dependent variable: "outgoing_interface"):
      SVM_Model <- svm(outgoing_interface ~ destination + TCP_
    throughput + MOS_score, data=Training_set, cost=cost_value
     , gamma=gamma_value)

      #Calculating predictions based on the Testing dataset:
```

```
      SVM_Prediction <- predict(SVM_Model, Testing_set, type="class")

      #Comparing prediction against original relapsed cases to find out number of matching:
      SVM_matched = 0
      for ( m in 1:nrow(Testing_set)){
          if (Testing_set$outgoing_interface[m] == SVM_Prediction[m]){
                SVM_matched = SVM_matched + 1
          }
      }

      #Calculating the Accuracy rate of each calculation and Storing it in Accuracy vector:
      SVM_Accuracy <- (SVM_matched * 100 ) / nrow(Testing_set)
      SVM_ACC_Values <- append( SVM_ACC_Values , SVM_Accuracy)

    }

    #Stroing Accuracy results of each of 9 different SVM structures in the matrix
    SVM_ACC_Matrix[count,1] <- cost_value
    SVM_ACC_Matrix[count,2] <- gamma_value
    SVM_ACC_Matrix[count,3] <- mean(SVM_ACC_Values)
    SVM_ACC_Matrix[count,4] <- max(SVM_ACC_Values)

    count <- count + 1
  }
}
```

DT algorithm is also used different configurations to train ML Model, for instance code listing **"DT algorithm"** shows the DT algorithm used of training and validating of four different DT models (four structures by four different values for "complexity parameter").

**Code listing: DT algorithm**

```
# Initiating an empty vector to hold calculated result of 30 iteration:
```

```r
DT_ACC_Values <- NULL

# Initiating a matrix to hold the Mean and Max accuracies of 4
    different DT models:
DT_ACC_Matrix = matrix(data=NA, nrow=4, ncol=3)
count = 1

#a "for" loop to define 4 different values of complexity
    parameter(cp):
for (cp_value in c(0.00001, 0.0001 , 0.001, 0.01)){

  #a "for" loop to perform "HoldOut Method" - 30 repetition:
  for (i in 1:30){

    #Dividing data into Training and Testing sets by random
    selection of rows from original dataset
    #(2/3 will be place into Training_Set and rest of them
    into Testing_Set)
    Training_rows <- sample(1:nrow(df), (2/3)*nrow(df) ,
    replace=F)
    Testing_rows <- setdiff(1:nrow(df), Training_rows)
    Training_set <- df[Training_rows,]
    Testing_set <- df[Testing_rows,]


    # Building a DT model based on training dataset (dependent
     variable: "outgoing_interface"):
    DT_Model <- rpart(outgoing_interface ~ destination + TCP_
    throughput + MOS_score, data=Training_set , control =
    rpart.control(cp = cp_value))

    #Calculating predictions based on the Testing dataset:
    DT_Prediction <- predict(DT_Model, Testing_set, type="prob
    ")
    DT_Prediction_df <- data.frame(DT_Prediction)
    for (n in 1:nrow(DT_Prediction_df)){
        if(max(DT_Prediction_df[n,c(1:3)]) == DT_Prediction_df
    $link_1[n]){
            DT_Prediction_df$result[n] = "link_1"
        } else if(max(DT_Prediction_df[n,c(1:3)]) == DT_
    Prediction_df$link_2[n]){
            DT_Prediction_df$result[n] = "link_2"
        } else if(max(DT_Prediction_df[n,c(1:3)]) == DT_
    Prediction_df$link_3[n]){
            DT_Prediction_df$result[n] = "link_3"
        }
    }

    #Comparing prediction against original relapsed cases to
    find out number of matching:
    DT_matched = 0
      for ( m in 1:nrow(Testing_set)){
          if (Testing_set$outgoing_interface[m] == DT_
    Prediction_df$result[m]){
              DT_matched = DT_matched + 1
          }
      }


    #Calculating the Accuracy rate of each calculation and
    Storing it in Accuracy vector:
    DT_Accuracy <- (DT_matched * 100 ) / nrow(Testing_set)
    DT_ACC_Values <- append( DT_ACC_Values , DT_Accuracy)

  }

#Stroing Accuracy results of each of 4 different DT structures
     in the matrix
DT_ACC_Matrix[count,1] <- cp_value
DT_ACC_Matrix[count,2] <- mean(DT_ACC_Values)
DT_ACC_Matrix[count,3] <- max(DT_ACC_Values)

count <- count + 1
}
```

## 3.4 Evaluation

As mentioned earlier, the key problem this work is going to address is to find the best outgoing link per each flow of inter-branch traffic and therefore to optimize the utilization of these critical links in an efficient way, using supervised ML. In addition to preprocessing of the data for the algorithm, in order to accurately and consistently test the performance measure of ML solution to the problem, following concerns were considered when selecting and using algorithms:

- Different techniques and ML algorithms selected to predict the same problem, two non-linear methods: Neural Network and SVM, and one Decision-Tree based classification method
- For each algorithm different structures applied by configuring various values for some of their key arguments (spot-check)
- Predictions made by a trained model on the test dataset - 66%–34% training/validation split
- Repeating training/validation split method and calculating the mean result of all repetitions (repeated holdout method)
- Using Accuracy as test metric for prediction results:
  *Correctly predicted (x) divided by total number of testing instances (y).*

Next step after validation of predictions, was to implement some real samples of user traffic flows on the network, and assess the core link utilizations and their routing efficiency before and after applying proposed policies (by the result of ML algorithms). For this purpose, four concurrent traffic flows initiated from HQ location toward Branch and DC networks, as listed in Table 10.

**Table 10.** User Traffic Flows

| **Flow ID** | **Source IP** | **Destination IP** | **Service** | **Protocol** | **Required MOS Score** | **Required TCP Throughput** | **Out. int.** |
|---|---|---|---|---|---|---|---|
| 18 | 172.16.7.18 (Headquarter) | 172.16.18.8 (Branch) | VoIP | RTP SIP H232 | 3.8 | 1920000 | NA |
| 28 | 172.16.7.28 (Headquarter) | 172.16.28.8 (Branch) | Biz _ Data | HTTPS POP3 SMTP | 3 | 6000000 | NA |
| 19 | 172.16.7.19 (Headquarter) | 172.16.19.9 (Datacenter) | Video Surveillance | RTP RTCP RTSP | 4 | 4000000 | NA |
| 29 | 172.16.7.29 (Headquarter) | 172.16.29.9 (Datacenter) | Backup Transfer | FTP WebDAV | 3 | 4000000 | NA |

Each traffic flow is representing real traffic used by different application or service, therefore each has different MoS and Throughput requirements:

| Flow_ID | Service | Representation of : |
|---|---|---|
| 18 | VoIP | 30 concurrent g.711 calls (64 Kbps) |
| 28 | Biz _ Data | 50 concurrent heavy user traffic (120 kbps) |
| 19 | Video Surveillance | 4 concurrent HD video channels |
| 29 | Backup Transfer | 5 GB of data transfer (over maximum 3 hours) |

Default routing policy for overlay network was based-on BGP updates from underlay network to edge nodes (R7, R8 and R9), mentioning only directly connected underlay nodes as next-hop for packet forwarding. In order to simulate Internet WAN connectivity, underlay network was configured to dynamically update routing paths. Therefore, after each update HQ edge node (R7) could receive different next hop address for the same destination network. In addition, another two flows (see Table 11) generated for the purpose of external (non-Biz) traffic simulation along with other flows mentioned above, which should be routed throughout least quality/Speed link. (In this case, Link 2):

**Table 11.** User Traffic to External destinations

| Flow ID | Source IP | Destination IP | Application | Protocol | MOS Score | TCP Throughput | Notes |
|---|---|---|---|---|---|---|---|
| 30 | 172.16.7.30 | 1.1.1.130 (External) | Internet call, Skype, ... | Skype UDP TCP | 3.7 | 2000000 | Group video Conf. |
| 31 | 172.16.7.31 | 1.1.1.131 (External) | Web Surfing, ... | P2P HTTP HTTPS | 3 | 2500000 | 50 light users (50 kbps each) |

Then, the required throughput of all six flows increased by 30 percent and tested on the network to simulate rush hours and peak times as well. Finally, already trained and validated ML models used to predict best outgoing interface for user flows, in both normal and peak time's situations. The results of ML prediction then used to build routing policies [23] to override default routing on

[23] 14. RFC-1102 : D. Clark, Policy Routing in Internet Protocols, `https://tools.ietf.org/html/rfc1102`, IETF. RFC 2475

edge node. For example, Code listing **"Prediction"** shows the proposed link for forwarding first flow (ID 18):

**Code Listing : Prediction**

```
> predict(ANN_Model, traffic_flows[1,], type="raw")
        link_1       link_2     link_3
1 0.003980481 4.14895e-06 0.9960154

> predict(SVM_Model, traffic_flows[1,], probability = TRUE)
       link_1      link_2     link_3
1 0.03189795 0.05071545 0.9173866

> predict(DT_Model, traffic_flows[1,], type="prob")
  link_1 link_2 link_3
1      0      0      1
```

Statistics once more gathered by NMS before and after applying these polices. Next section describes the comparison of resulting statistics and experimental observations.

## 4 Experimental results

To undertake a qualitative evaluation of proposed methodology, R software suite (version 3.4.0) was employed to evaluate the three utilized supervised machine-learning algorithms. Each model was trained with (2/3) of samples and then validated with (1/3) samples, both randomly selected from dataset and then tested with some real flows. Different structures combined in four different configuration sets for each algorithm:

– ANN: four configuration sets, each with three different values for "neuron_size" argument and three different values for "max_itr" argument ( = nine structure sets per each configuration set)
– SVM: four configuration sets, each with three different values for "Cost" argument and three different values for "Gamma" argument ( = nine structure sets per each configuration set)
– DT: four configuration sets, each with four different values for the argument called "complexity parameter (cp)" ( = four structure sets per each configuration set)

Figure 9 demonstrates how these structures are logically nested.

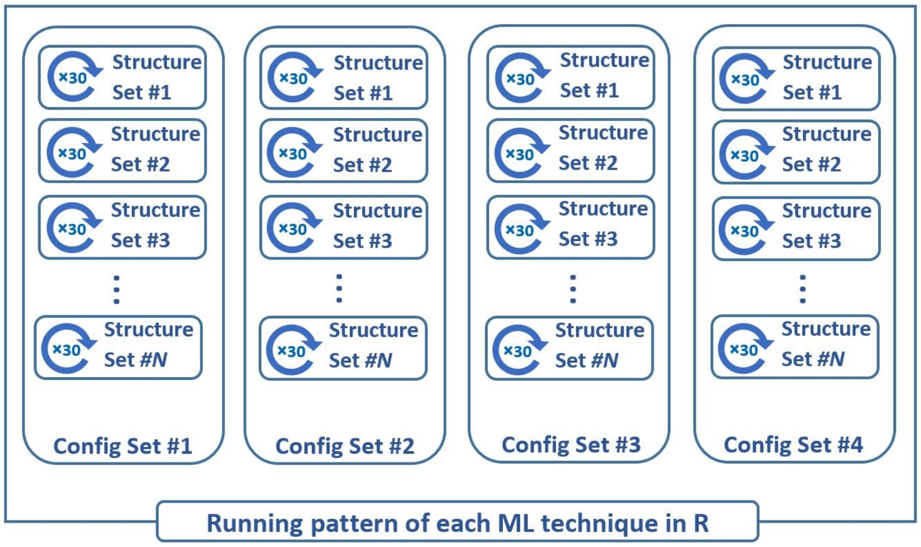


**Fig. 9.** Different Structure Sets for each algorithm

## 4.1 Accuracy

After executing each structure set (each structured set is also repeated 30 times, therefore 30 different training and validation sets for each structure set), the mean and max accuracy results of each 30 iterations gathered, as listed in Tables 12, 13 and 14 for ANN, SVM and DT algorithms, respectively.

As seen from table 12 and figure 10, the mean accuracy of each 30 iteration for ANN, grows as the values of neuron_size and maxi_itr arguments increase. With the values of 100 and 1300 for neuron_size and maxi_itr respectively, mean accuracy is above the 90%. Meantime, it seems that the max accuracy is almost high between all iterations, which means there is always some high accurate models, even with the low values of neuron_size and maxi_itr arguments.

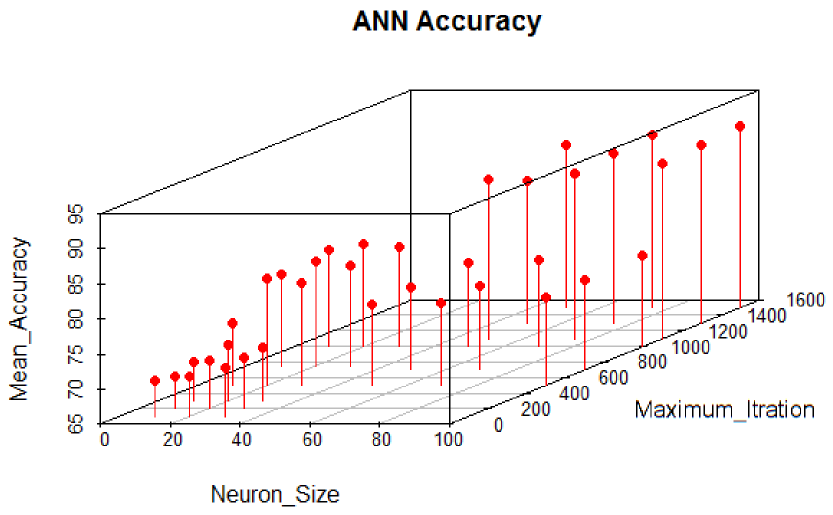


**Fig. 10.** ANN Accuracy graph

In the case of SVM algorithm, the accuracy growth is not smooth as ANN. However, also shows (as seen from Table 13 and Figure 11) mean accuracy grows especially with the high value of gamma argument, as more accurate models begin with Gamma value of 1 or higher.

**Table 12.** ANN Accuracy result

| **Structure set** | **Neuron Size** | **Maximum Itration** | **Mean Accuracy** | **Max Accuracy** |
|---|---|---|---|---|
| 1 | 10 | 100 | 70.15189 | 93.47291 |
| 2 | 10 | 200 | 69.52176 | 94.70443 |
| 3 | 10 | 300 | 70.58566 | 95.93596 |
| 4 | 20 | 100 | 70.64758 | 95.93596 |
| 5 | 20 | 200 | 71.8867 | 95.93596 |
| 6 | 20 | 300 | 73.02887 | 95.93596 |
| 7 | 30 | 100 | 71.87837 | 95.93596 |
| 8 | 30 | 200 | 72.23933 | 95.93596 |
| 9 | 30 | 300 | 72.57115 | 95.93596 |
| 10 | 10 | 500 | 73.91626 | 94.58128 |
| 11 | 10 | 750 | 77.97414 | 95.68966 |
| 12 | 10 | 1000 | 78.87521 | 95.68966 |
| 13 | 20 | 500 | 80.18062 | 95.68966 |
| 14 | 20 | 750 | 79.96716 | 95.68966 |
| 15 | 20 | 1000 | 79.61754 | 95.68966 |
| 16 | 30 | 500 | 79.57776 | 95.68966 |
| 17 | 30 | 750 | 79.31496 | 95.68966 |
| 18 | 30 | 1000 | 79.24375 | 96.05911 |
| 19 | 50 | 500 | 76.57635 | 96.18227 |
| 20 | 50 | 700 | 76.85482 | 96.18227 |
| 21 | 50 | 1000 | 76.97001 | 96.18227 |
| 22 | 70 | 500 | 76.84062 | 96.18227 |
| 23 | 70 | 700 | 76.9757 | 96.18227 |
| 24 | 70 | 1000 | 77.27311 | 96.18227 |
| 25 | 100 | 500 | 77.54132 | 96.18227 |
| 26 | 100 | 700 | 77.7404 | 96.18227 |
| 27 | 100 | 1000 | 77.95554 | 96.18227 |
| 28 | 50 | 1100 | 87.79146 | 98.02956 |
| 29 | 50 | 1300 | 85.37767 | 98.02956 |
| 30 | 50 | 1500 | 88.12261 | 98.02956 |
| 31 | 75 | 1100 | 88.50164 | 98.02956 |
| 32 | 75 | 1300 | 89.21346 | 98.02956 |
| 33 | 75 | 1500 | 89.66749 | 98.02956 |
| 34 | 100 | 1100 | 89.97654 | 98.76847 |
| 35 | 100 | 1300 | 90.34996 | 98.76847 |
| 36 | 100 | 1500 | 90.79183 | 98.7684 |

**Table 13.** SVM Accuracy results

| **Structure set** | **gamma_value** | **cost_value** | **Mean Accuracy** | **Max Accuracy** |
|---|---|---|---|---|
| 1 | 0.001 | 10 | 52.47537 | 64.03941 |
| 2 | 0.01 | 10 | 58.47906 | 67.48768 |
| 3 | 0.1 | 10 | 62.10591 | 88.91626 |
| 4 | 0.001 | 100 | 60.37048 | 88.91626 |
| 5 | 0.01 | 100 | 61.48358 | 88.91626 |
| 6 | 0.1 | 100 | 66.38136 | 93.84236 |
| 7 | 0.001 | 500 | 66.211 | 93.84236 |
| 8 | 0.01 | 500 | 68.16041 | 93.84236 |
| 9 | 0.1 | 500 | 71.02132 | 93.84236 |
| 10 | 0.001 | 700 | 65.16831 | 67.85714 |
| 11 | 0.01 | 700 | 76.97455 | 93.71921 |
| 12 | 0.1 | 700 | 82.23727 | 94.21182 |
| 13 | 0.001 | 850 | 77.97003 | 94.21182 |
| 14 | 0.01 | 850 | 80.34154 | 94.21182 |
| 15 | 0.1 | 850 | 82.39943 | 94.21182 |
| 16 | 0.001 | 1000 | 79.90676 | 94.21182 |
| 17 | 0.01 | 1000 | 81.2187 | 94.21182 |
| 18 | 0.1 | 1000 | 82.51095 | 94.33498 |
| 19 | 1 | 10 | 91.74877 | 93.34975 |
| 20 | 10 | 10 | 92.63752 | 95.5665 |
| 21 | 100 | 10 | 92.89819 | 95.68966 |
| 22 | 1 | 100 | 92.95259 | 95.68966 |
| 23 | 10 | 100 | 93.12315 | 95.68966 |
| 24 | 100 | 100 | 93.18692 | 95.68966 |
| 25 | 1 | 500 | 93.34975 | 95.68966 |
| 26 | 10 | 500 | 93.42826 | 95.68966 |
| 27 | 100 | 500 | 93.42182 | 95.68966 |
| 28 | 1 | 700 | 93.96141 | 95.5665 |
| 29 | 10 | 700 | 93.90189 | 95.68966 |
| 30 | 100 | 700 | 93.73974 | 95.68966 |
| 31 | 1 | 850 | 93.79516 | 95.68966 |
| 32 | 10 | 850 | 93.79639 | 95.68966 |
| 33 | 100 | 850 | 93.70553 | 95.68966 |
| 34 | 1 | 1000 | 93.77199 | 95.68966 |
| 35 | 10 | 1000 | 93.75667 | 95.68966 |
| 36 | 100 | 1000 | 93.65672 | 95.68966 |

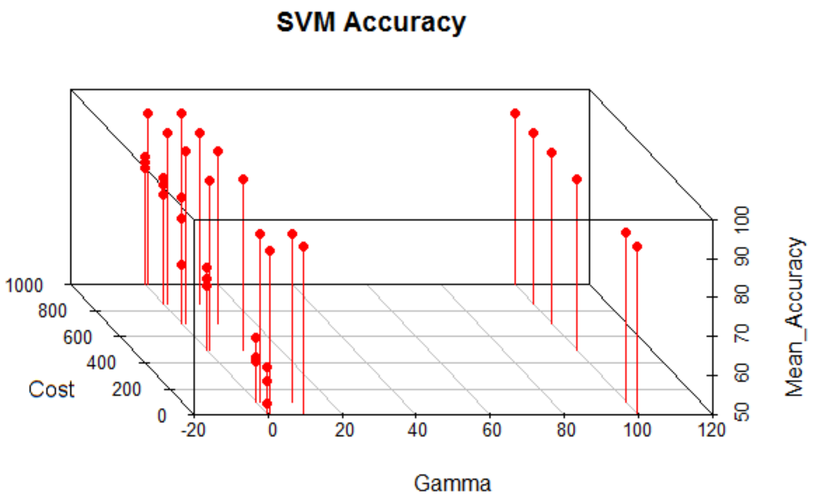


**Fig. 11.** SVM Accuracy graph

Complexity parameter (cp) has significant effect of the result of DT classification. Sixteen different values of used while training DT model, and as table 14 and figure 12 are showing, lower values of complexity parameter resulted in high accurate DT models to be used for prediction. It shows high accurate models until the ”cp value” of 0.001, but accuracy drops after this value, significantly.

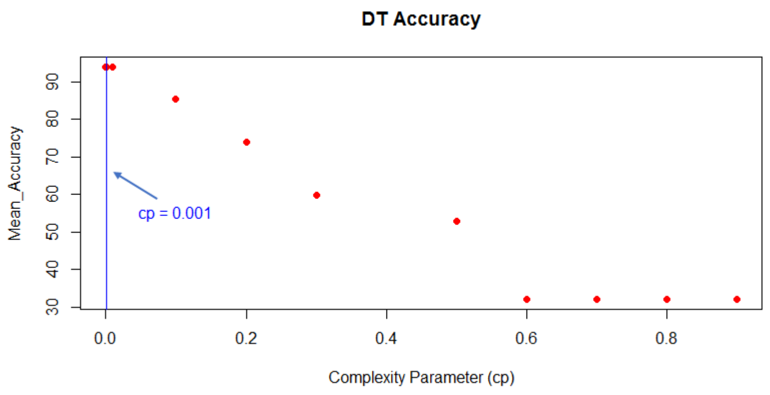


**Fig. 12.** DT Accuracy graph

**Table 14.** DT Accuracy results

| Structure set | cp_value | Mean_Accuracy | Max_Accuracy |
|---|---|---|---|
| 1 | 0.000000001 | 94.04762 | 95.68966 |
| 2 | 0.00000001 | 93.867 | 95.68966 |
| 3 | 0.0000001 | 93.91352 | 95.68966 |
| 4 | 0.000001 | 93.92652 | 95.93596 |
| 5 | 0.00001 | 94.04762 | 95.19704 |
| 6 | 0.0001 | 93.89368 | 95.19704 |
| 7 | 0.001 | 94.02573 | 95.3202 |
| 8 | 0.01 | 93.8906 | 95.3202 |
| 9 | 0.1 | 85.45977 | 93.47291 |
| 10 | 0.2 | 73.87315 | 93.47291 |
| 11 | 0.3 | 59.922 | 93.47291 |
| 12 | 0.5 | 52.97824 | 93.47291 |
| 13 | 0.6 | 32.22496 | 33.25123 |
| 14 | 0.7 | 32.22085 | 33.25123 |
| 15 | 0.8 | 32.13191 | 33.25123 |
| 16 | 0.9 | 32.06383 | 33.25123 |

In addition to the accuracy of trained models in three algorithms, their run-time is also checked by comparing system time before and after running the algorithm as seen in code listing **"Prediction"**:

**Code Listing : Prediction**

```
start.time <- Sys.time()
#... omitted lines ... Relevant codes of the ML algorithm...
end.time <- Sys.time()
time.taken <- end.time - start.time
```

ANN had the significantly high runtime of 1.394665 hours, while SVM and DT algorithms ran much faster and they executed in 6.437531 minutes and 6.809009 seconds, respectively. Despite the high runtime duration, the highest accuracy resulted by ANN, at the value of 98.76847%.

## 4.2 Traffic Analyze

In order to check link utilization, first, testing traffic flows initiated from HQ toward other branches as well as external destinations (for reference see tables 10, 11, and figure 4), to be forwarded via default routing mechanism. As Figure 13 shows, total amount 20.4 Mbps of traffic was receiving at HQ edge router (incoming) to be forwarded via three edge links. Live traffic statistics gathered via SNMP protocol from R7, by NMS appliance.

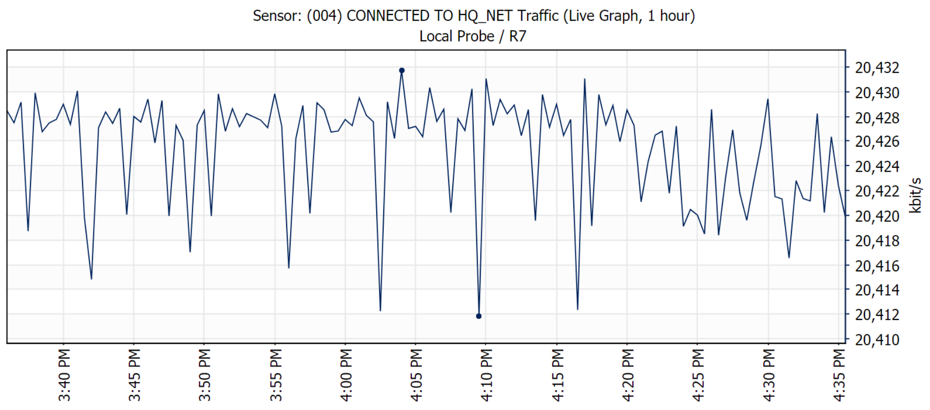


**Fig. 13.** Incoming Traffic from HQ to R7 edge router

Figures 14, 15 and 16 show the out-bound traffic graphs for link 1, link 2 and link 3. Wireshark software used to capture the flows forwarded out by each link and result listed in table 15. Due to the underlay infrastructure and link speeds (see figure 4 for reference), shortest path for reaching at Datacenter and Branch locations are via link 1 and 3 respectively. Link 2 has lower speed and longer paths to both locations, making it less suitable to be used for critical data transfer, but appropriate for non-biz data (e.g. web surfing). However, HQ edge router has no information of underlay paths and it uses advertised routing metrics of underlay edge nodes to forward traffic.

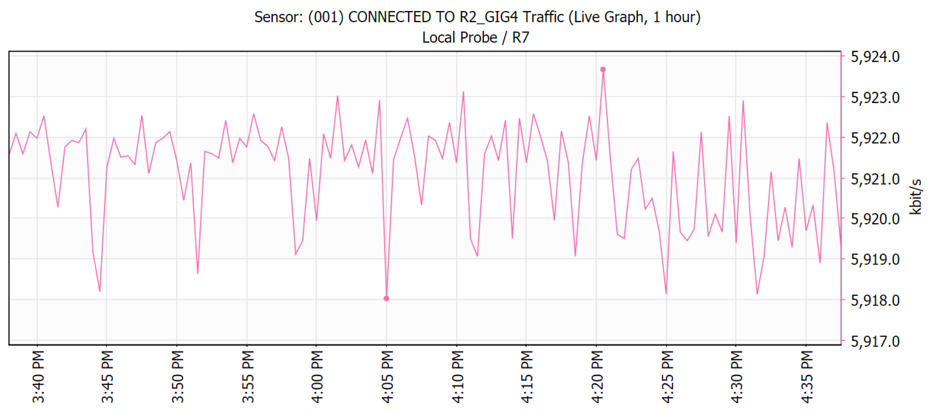


**Fig. 14.** Outbound Traffic link Utilization, Link 1

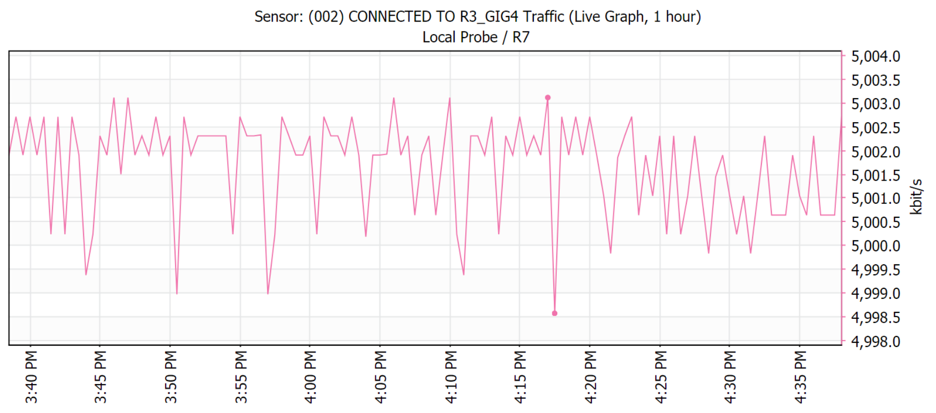


**Fig. 15.** Outbound Traffic link Utilization, Link 2

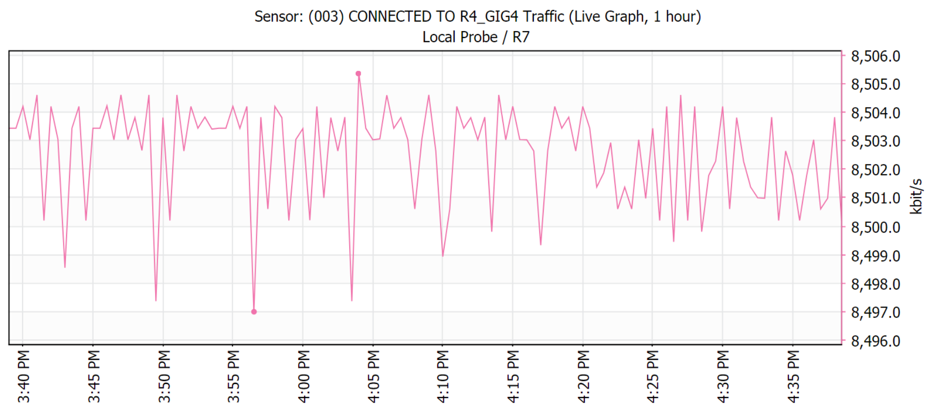


**Fig. 16.** Outbound Traffic link Utilization, Link 3

**Table 15.** Packet Capture of user traffic flows, before ML

| **Edge Link** | **Time** | **Source IP** | **Destination IP** | **Protocol** | **Port numbers** | **Length** |
|---|---|---|---|---|---|---|
| Link 1 | 6/14/2017 16:06 | 172.16.7.19 | 172.16.19.9 (Datacenter) | UDP | 13174 → 554 | 1412 |
| | 6/14/2017 16:06 | 172.16.7.18 | 172.16.18.8 (Branch) | UDP | 32768 → 16384 | 160 |
| Link 2 | 6/14/2017 16:15 | 172.16.7.28 | 172.16.28.8 (Branch) | TCP | 25164 → 443 | 1460 |
| Link 3 | 6/14/2017 16:33 | 172.16.7.29 | 172.16.29.9 (Datacenter) | TCP | 45123 → 21 | 1420 |
| | 6/14/2017 16:33 | 172.16.7.30 | 1.1.1.130 (External) | TCP | 443 → 34625 | 1350 |
| | 6/14/2017 16:33 | 172.16.7.31 | 1.1.1.131 (External) | TCP | 40435 → 80 | 1460 |

Result shows less efficient routing and link utilization. For example, Link 1 (10,000 Kbps) is utilized almost by 60%, leaving 40% unused, and only half of that traffic (30%) uses shorter path to the DC, while other 30% is utilized by the traffic goes to Branch office (longer distance, therefore less efficient routing). Captures also show that link 2 (5,000 Kbps, unstable link) is fully utilized by critical business data flows (longer path to Branch office destination, means more packet loss and non-efficient routing). Figure 16 shows utilization of 80% for link 3 (10,000 Kbps). However, by checking the packet capture results in table 15, it shows that half the utilization is due to non-biz data and other half of it is due to the traffic goes to Datacenter destination (longer path), meaning less optimized link utilization.

After applying policies generated by the results of ML prediction for appropriate outgoing interface for user traffic flows, link utilization statistics also gathered in a live method by SNMP, as well as packet capture for identifying traffic flows of each link. Figures 17, 18 and 19 are the plotting results of link utilization for link 1, 2 and 3 respectively. And packet capturing results have listed in table 16.

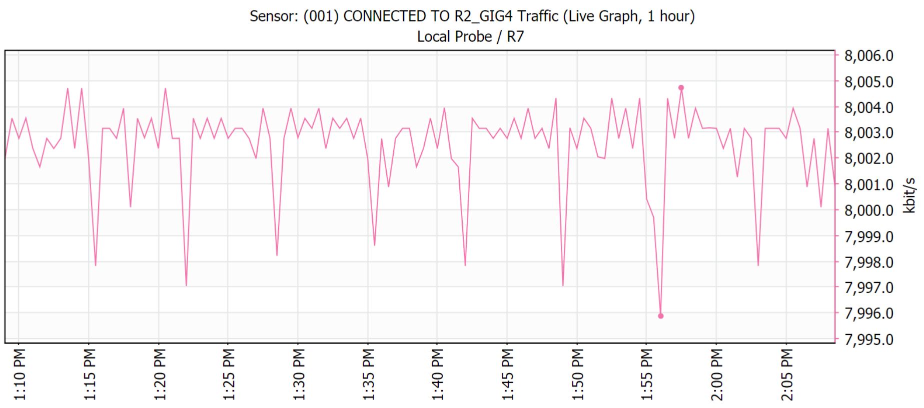


**Fig. 17.** Outbound Traffic link Utilization, Link 1, After ML policy

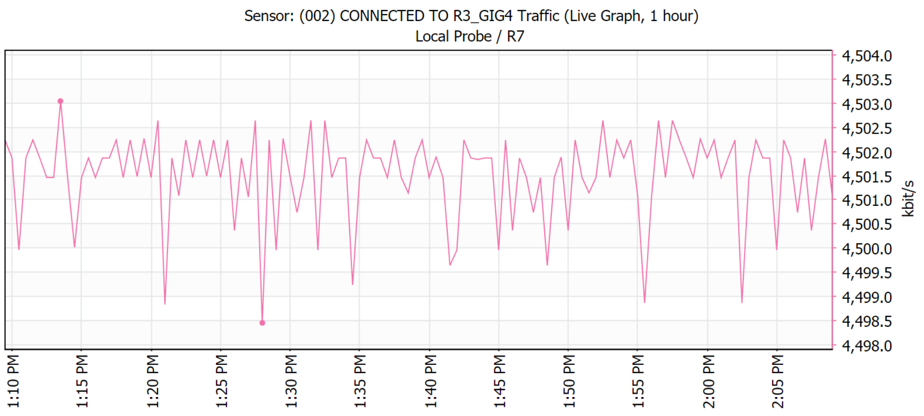


**Fig. 18.** Outbound Traffic link Utilization, Link 2, After ML policy

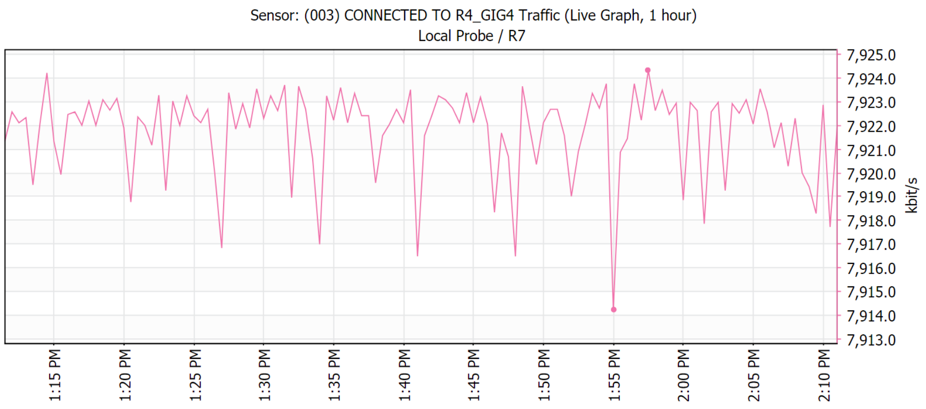


**Fig. 19.** Outbound Traffic link Utilization, Link 3, After ML policy

**Table 16.** Packet Capture of user traffic flows, After ML policy

| Edge Link | Time | Source IP | Destination IP | Protocol | Port numbers | Length |
|---|---|---|---|---|---|---|
| Link 1 | 6/15/2017 13:52 | 172.16.7.19 | 172.16.19.9 (Datacenter) | UDP | 13174 → 554 | 1412 |
|  | 6/15/2017 13:52 | 172.16.7.29 | 172.16.29.9 (Datacenter) | TCP | 45123 → 21 | 1474 |
| Link 2 | 6/15/2017 14:04 | 172.16.7.130 | 1.1.1.130 (External) | TCP | 443 → 34625 | 1404 |
|  | 6/15/2017 14:04 | 172.16.7.131 | 1.1.1.130 (External) | TCP | 40435 → 80 | 1460 |
| Link 3 | 6/15/2017 14:08 | 172.16.7.18 | 172.16.18.8 (Branch) | UDP | 32768 → 16384 | 160 |
|  | 6/15/2017 13:08 | 172.16.7.28 | 172.16.28.8 (Branch) | TCP | 25164 → 443 | 1460 |

Significant differences are observed after applying policies, in which all three links showing better utilization (80%, 90% and 79% for link 1, 2 and 3 respectively) with correct traffic. Also, traffic flows taking and shortest path to their direction, which means the link utilizations are now more optimized, while having efficient routing.

In order to simulate rush hours and peak times the required throughput of all six flows increased by 30 percent (see Table 17) and tested on the network, and again their statistics gathered before and after ML policies for observation and comparison.

**Table 17.** User Flows with 30% in Traffic throughput

| Flow ID | Application / Service | Throughput (Kbps) + 30% |
|---|---|---|
| 18 | VoIP | 2496000 |
| 28 | Biz _ Data | 7800000 |
| 19 | Video Surveillance | 5200000 |
| 29 | Backup Transfer | 5200000 |
| 30 | Internet call / Skype | 2600000 |
| 31 | Web Surfing | 3250000 |

A live graph of total traffic received at HQ's edge router (R7) illustrated in figure 20. It shows the amount of 26.54 Mbps for all flows combined, which is more than the outgoing capacity of all three links (10Mbps + 5Mbps + 10Mbps = 25Mbps).

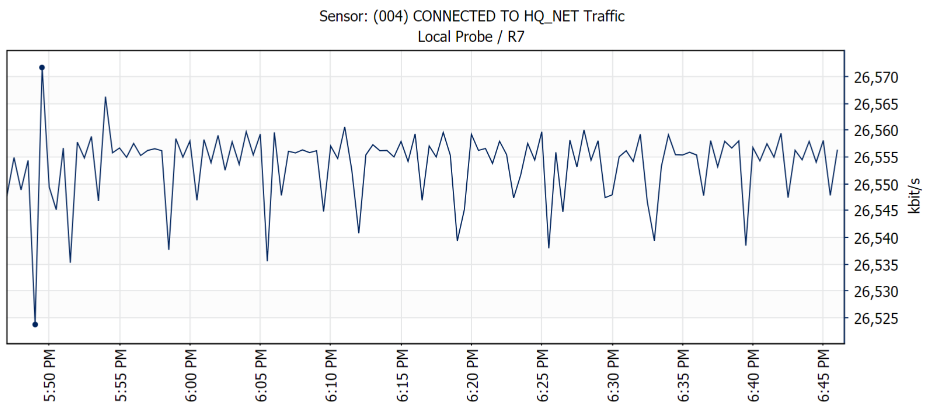


**Fig. 20.** Inbound Traffic received from HQ into R (peak time)

Therefore, to prevent flow packets being dropped, traffic-shaping techniques were implemented to put the them in forwarding queue based on their priority settings. Figure 21 shows the Peak hour's utilization graphs for each link, before and after ML policies.

Results showed that in addition to forwarding traffic flows via most appropriate outgoing interface after apply policies, also all three links have utilized to their upper capacity. While before, although routing was not efficient as some flows were taking longer and unstable paths, even link 1 was not fully utilized to it upper limit, which means rather more amount of traffic were dropping.

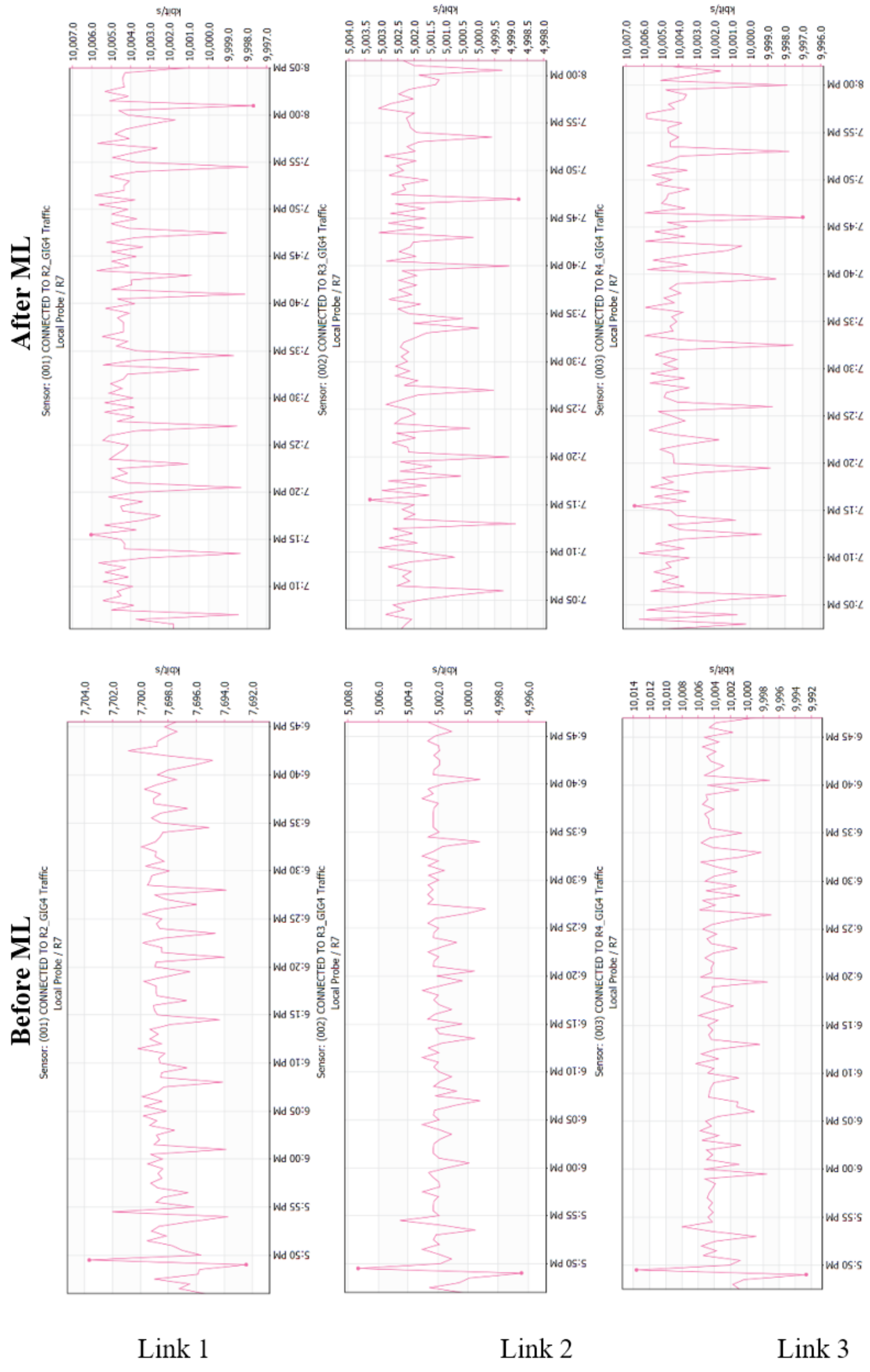


**Fig. 21.** Outbound link Utilization, before and after ML (peak time)

# 5 Conclusions

The present work used three supervised machine learning techniques for prediction of the best outgoing link on a per-flow basis, solely using IP stream attributes derived and gathered from packets in a roundtrip measurement method. Following the steps of proposed methodology, first flow statistics and metrics collected continuously for almost 3 days and thereafter summarized into training and testing feature sets based on influential attributes, to train and validate ML models of ANN, SVM and DT algorithms. Prediction of the most appropriate outbound interface for traffic management on critical links conducted using the flow destination, TCP throughput and MoS score measurements.

Different configuration of different structure of each algorithm tested several times, where for ANN the prediction accuracy increased as the values of "Maximum Iteration and "Neuron size" attributes were grew up. The same happened to SVM as the accuracy of its models increased while using higher values for "Gamma" and "Cost" attributes. In DT accuracy for lower values of "Complexity Parameter (CP)" was significantly high, however a sudden decrease of accuracy happened to those DT models with the "CP" value of more than 0.01. Meantime even though the ANN algorithm had a very long runtime period of 1.394665 hours (in comparison to two other algorithms), it showed the highest accuracy (96% and above) per structure sets with the Neuron Size of 50 or more and Maximum Iteration of 700 or more per model.

Models then tested on real user flows and as result appropriate routing policy applied on the edge router. Before this policy edge router was using its default routing protocol for packet forwarding, which despite an inappropriate routing of flows, the utilization of links was also inefficient. Link 1 was utilized almost 60%, leaving 40% unused, and only half of the utilization (30%) were due to appropriate flow routing. Although the link2 (unstable link) was fully utilized but it was due to business data, which obviously resulted into high loss rate in critical flows. And as observed on link 3, it was highly dominated by non-critical flows. Significant improvement in link utilization observed after applying ML policy, where stable links ( link 1 and 3) were only utilized by critical flows and unstable link (link 2) was used for forwarding less important data. These also resulted an efficient routing as significantly lower loss rates observed. Policy also applied to the traffic on peak time and rush hour situation and again meaningful improvement in load distribution and loss rate observed.

An important capability for flow measurement in live deployments is to detect when there are significant estimation errors. Although this work studied on real networking infrastructure, however the scale of testing environment was relatively small therefore smaller amount of changes in network could happen. A direction for future works can be studies on larger environments where errors could be indicative of a large number of changes in network routing, causing an abrupt change in delay, loss, and throughput. Since this present study only considered predictors trained and tested on the same path, a number of features, such as the length of a path and MTU, are implicit. This resulted in simpler use of ML algorithms for path-based prediction, because some of packet forwarding details that would be required, can be hidden from the user. Another potential challenge for future researches is identifying and measuring all factors – path properties, TCP/UDP flavors, operating system parameters – that might affect throughput and other flow quality metrics.

## References

1. Thuy T.T. Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*, 10(4), 2008.
2. Abu Alsheikh Mohammad, Lin Shaowei, Niyato Dusit, and Tan Hwee-Pink. Machine learning in wireless sensor networks: Algorithms, strategies, and applications. *IEEE COMMUNICATIONS SURVEYS AND TUTORIALS*, 16(4):1996–2018, 2014.
3. Grzegorz Kolaczek Jaroslaw Bernacki. Anomaly detection in network traffic using selected methods of time series analysis. *International Journal of Computer Network and Information Security(IJCNIS)*, 7(9), aug 2015.
4. Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Inf. Sci.*, 177(18):3799–3821, September 2007.
5. Hongli Zhang, Gang Lu, Mahmoud T. Qassrawi, Yu Zhang, and Xiangzhan Yu. Feature selection for optimizing traffic classification. *Comput. Commun.*, 35(12):1457–1471, July 2012.
6. Jun Zhang, Xiao Chen, Yang Xiang, Wanlei Zhou, and Jie Wu. Robust network traffic classification. *IEEE/ACM Trans. Netw.*, 23(4):1257–1270, August 2015.
7. T. Auld, A. W. Moore, and S. F. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, Jan 2007.
8. Umer Khan, Lars Schmidt-Thieme, and Alexandros Nanopoulos. Collaborative svm classification in scale-free peer-to-peer networks. *Expert Systems With Applications*, 69(Complete):74–86, 2017.
9. Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3):67–82, July 1997.
10. Paul Barford and Mark Crovella. Critical path analysis of tcp transactions. *SIGCOMM Comput. Commun. Rev.*, 30(4):127–138, August 2000.
11. Vern Paxson. End-to-end internet packet dynamics. *SIGCOMM Comput. Commun. Rev.*, 27(4):139–152, October 1997.
12. Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. On the characteristics and origins of internet flow rates. *SIGCOMM Comput. Commun. Rev.*, 32(4):309–322, August 2002.
13. Yin Zhang and Nick Duffield. On the constancy of internet path properties. pages 197–211, 2001.
14. V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, August 1988.
15. J¨org Ott and Dirk Kutscher. A disconnection-tolerant transport for drive-thru internet environments. *In The 24th Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM, Miami, FL,*, mar 2005.

16. Teunis J. Ott, J. H. B. Kemperman, and Matt Mathis. The stationary behavior of ideal tcp congestion avoidance, aug 1996.
17. Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. A stochastic model of tcp/ip with stationary random losses. *SIGCOMM Comput. Commun. Rev.*, 30(4):231–242, August 2000.
18. Roch Guerin Mukul Goyal and Raju Rajan. Predicting tcp throughput from non-invasive network sampling. *In The 21st IEEE Joint Conference of the IEEE Computer and Communications Societies, INFOCOM, New York, NY, USA*, jun 2002.
19. Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, 28(4):303–314, October 1998.
20. Martin Arlitt, Balachander Krishnamurthy, and Jeffrey C. Mogul. Predicting short-transfer latency from tcp arcana: A trace-based validation. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 19–19, Berkeley, CA, USA, 2005. USENIX Association.
21. Dong Lu, Yi Qiao, Peter A. Dinda, and Fabián E. Bustamante. Characterizing and predicting TCP throughput on the wide area network. In *25th International Conference on Distributed Computing Systems (ICDCS 2005), 6-10 June 2005, Columbus, OH, USA*, pages 414–424, 2005.
22. Martin Swany and Rich Wolski. Multivariate resource performance forecasting in the network weather service. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, SC '02, pages 1–10, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.
23. M. Mirza, J. Sommers, P. Barford, and X. Zhu. A machine learning approach to tcp throughput prediction. *IEEE/ACM Transactions on Networking*, 18(4):1026–1039, Aug 2010.