COMPARISON AND EVALUATION OF TRADITIONAL AND MODERN NETWORK DATA PATH PROTOCOLS

Waleed T. Al-Sit
Department of Computer Engineering, Mu'tah University Al-karak , Jodran
w_sitt@hotmail.com

ABSTRACT*— Recently, many new network paths were introduced, while old paths are still in use. The trade-offs remain vague and should be further addressed. Since the last decade, the Internet is playing a major role in persons' lives, and the demand on the Internet in all fields increased rapidly. To get a fast and secure connection to the Internet, the networks providing the service should get faster and more reliable. Many network data paths have been proposed to achieve the previous objectives since the 1970s. It started with the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) and later followed by several more modern paths, including the Quick UDP Internet Connections (QUIC), the Remote Direct Memory Access (RDMA), and the Data Plane Development Kit (DPDK). The question on which data path should be adapted and based on which features is occurred. In this work, we will try to answer this question using different perspectives such as the protocol techniques, latency and congestion control, the achieved throughput, middle boxes consideration, loss recovery mechanisms, developer productivity, and host resource utilization.*

Index Terms— Data path, TCP, UDP, QUIC, RDMA, DPDK.

I. INTRODUCTION

The availability of electronic appliances that store various types of data, or can hold desired services, lead to the need to share data or services with other electronic appliances. Computer networks, for example, facilitate the sharing of different types between computers, laptops, or any smart devices that can connect to the system. Computing resources can also be shared through a network such as printing a document on a printer that is shared through a network or uses remote shared storage. Secured sharing is also possible through networks connecting devices/users with different privileges. Although devices in different networks are connected in similar ways, computer networks differ according to the medium carrying the signals, the transmission protocols, the size, the topology, and the intent of the organization.

Since the last decade, the largest known computer network in the world is the Internet. The Internet plays a vital role, almost in all aspects of human life. In general, the workload in network systems has been overwhelmed. Therefore, network applications exploded in all fields to support the increasing demands of leveraging the networks. The network model is divided into seven logical layers to make the architecture of the network clearer, traceable, and to provide consistent and reliable services. Each layer performs a specific functionality.

Our work focuses mainly on the processes of the fourth layer, which is called the Transport layer, and occasionally we will explain related procedures from other layers where needed.

The primary function of the transport layer is to ensure packets are error-free prior transmission and that all of them arrive and correctly reassembled at the destination [1]. More specifically, the transport layer allows processes to communicate, and it controls the process-to-process delivery. The application data at the transport layer appears in the forms of segments that are ready for transmission, and all parts will be reassembled at the destination upon arrival [1]. The protocols functioning in the transport layer were classified according to the connection into connection-oriented or connectionless. This protocol starts by establishing a virtual connection before transferring segments from the source to the destination and terminates the connection once all data is transferred. Such protocols normally use specific mechanisms to control the flow and possible transmission errors. On the other hand, connectionless protocols do not establish connections before sending the data [2], and every single segment is treated as an independent packet [3, 4, 5]. The transferred data using such protocols are delivered to the destination process blindly without any flow or error control mechanisms.

The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) are the best-known examples for connection-oriented and connectionless protocols. TCP and UDP are approved to work in the transport layer, although they handle data transmission differently [6]. TCP uses a connection-oriented protocol that provides a very reliable mechanism of handling messaging or information transmission to guarantee message delivery. If any error occurred during transmission, the packet would be automatically re-sent over the network [7]. However, UDP employs a simpler transport model with a minimum of protocol technique. Additionally, using UDP, it is possible to transmit messages indicated as datagrams, voices, and videos with various internet services such as emails, webpages, video streaming, etc.

At the hardware side, network appliances based on dedicated hardware are also essential to the network system has a major effect on performance and reliability. The main task of the appliances is to forward and filter packets. Still, they support many other network functions (NFs), including quality of service (QoS), deep packet inspection (DPI), data encryption service, and so on [8]. However, hardware-based network appliances have some drawbacks. First, hardware-based network appliances are relatively expensive. The purchasing and power costs are unaffordable for a huge number of these appliances. Second, the management of hardware-based network appliances is somehow difficult as the administrator must learn different network operating system for the devices offered by various vendors of the appliances.

Additionally, the types of middleboxes are increasing day-after-day, which makes it more difficult for an administrator to master their new operating systems. Third, the hardware-based network appliances are space consuming and have poor flexibility. Finally, their development is slow, and it is years to catch up with the development of new network technologies [8].

In 2012, many large carriers proposed the concept of Network Function Virtualization (NFV) [9], to decouple the NF from the dedicated hardware appliances. This approach enabled shifting the NFs to software running on off-the-shelf servers. However, the move from hardware to software was not easy to progress. The huge obstacle was the gab in software performance compared to the hardware performance. Although Unix-based software routers provide high flexibility and a low cost, the packet processing was slow, and the vast delay was unacceptable for the network system. Therefore, many projects concluded that standard software is not suitable for highspeed packet forwarding scenario [9, 10, 11]. However, various research groups analyzed the bottleneck of network performance by building models to leverage the available network data paths or develop better ones. A network data path describes the set of functional units that perform the packet processing operations.

At the Intel side, a high-speed packet processing framework named Data plane Development Kit (DPDK) [12, 13] that creates a lower layer to perform all tasks such as allocating and managing memory for network packets in addition to buffering the needed packet descriptors and pass them through the Network Interface Card (NIC). Similarly, the Remote Dynamic Memory Access technology (RDMA) technology allows writing/read data from/to known memory regions of other network machines without involving their CPUs aiming to achieve high-speed data transfer. The last protocol involved in this work is the Quick UDP Protocol (QUIC), which is a Google, proposed encrypted, and multiplexed transport protocol. QUIC was implemented to improve HTTPS performance smoothly.

In this work, we try to answer the question of which data path should be adapted in which application. The trade-offs between the traditional data paths such as TCP and UDP compared to more modern data paths such as QUIC, RDMA, and DPDK, still need to be further studied. The rest of this work is structured as follows: Section 2 presents the used terms. Section 3 presents an overview of network data paths. A thorough comparison between the studied network data paths is in Section 4, and the conclusion can be found in the last section.

II. TERMS

A. Flow Control

Flow control is the mechanism that is used to make sure the sent proportion is within the receiving capabilities of the receiver. If the received data is higher than the receiver's capabilities, an overflow problem occurs, causing the part of the transferred data to be lost and need to be retransmitted [14].

B. Congestion Control

It occurs if a network node or a link is carrying more data than it can handle. The congestion problem might cause packet losses, new connections to be blocked, or merely a delay in the transmission [15].

C. Error Control

It is the technique used for the detection and correction of data blocks during the communications. More specifically, error control is used to checks how strong the characters are at the bit and packet levels to ensure that the received data is identical to the transmitted data [16].

D. Latency

Latency measures the time it takes for some data to get to its destination over the network. It is usually measured as a round trip time (RTT) - the time taken for information to get to its destination and back again [17].

E. Multiplexing and Demultiplexing

Multiplexing is the process used to combine and transmit multiple data streams over a single data channel, although the data might come from different sources. Additionally, multiplexing helps network devices to communicate with other network devices without having to dedicate a specific connection for each communication. At the destination side, the multiplexed signal must be separated again using a process called demultiplexing [18].

F. Network Quality of Service

Quality of Service (QoS) refers to the capability of a network to provide better service to selected network traffic over various technologies, including Frame Relay, Ethernet, 802.1 networks, and IP-routed networks that may use any or all these underlying technologies. The primary goal of QoS is to provide priority, including dedicated bandwidth, controlled jitter, and latency, delay, packet loss parameters. Also important is making sure that providing priority for one or more flows does not make other streams fail [19].

G. Retransmission

It is a mechanism used by protocols to ensure reliable transmission where a lost or corrupted signal must be transmitted again [20].

III. NETWORK DATA PATH

In this section we will present the data paths starting the traditional ones followed by the most modern ones.

A. The Transmission Control Protocol

The Transmission Control Protocol (TCP) (RFC 0793) is the most important protocol in the Internet protocol suite and considered the cornerstone protocol in the transport layer [1]. TCP was initiated during the early network implementation stages to complement the Internet Protocol (IP). Therefore, the common name for the suite is TCP/IP.

TCP is a connection-oriented point to point protocol where the "handshake" is required before one application process can start sending data to another application. The handshake process means that applications must exchange some preliminary segments to establish the parameters of the

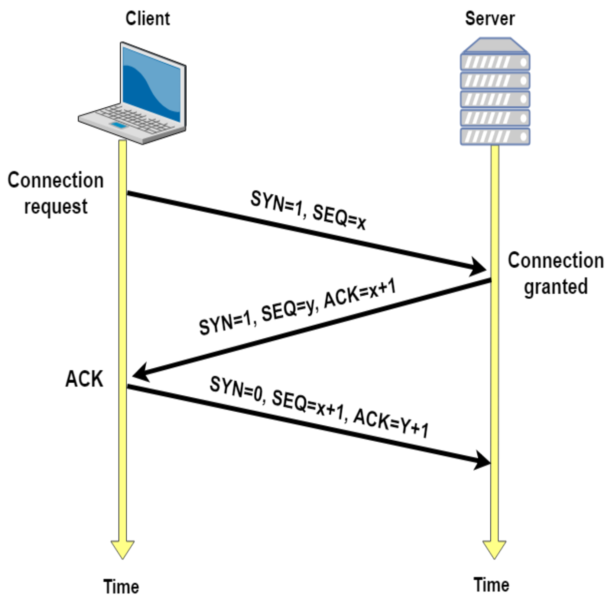

Figure 1: three-way handshake in TCP: segment exchange.

ensuing data transfer [18]. Therefore, TCP service is considered reliable, ordered, and error-checked. TCP was optimized for accurate delivery, not the timely delivery. Thus,
long delays might occur waiting for out-of-order messages or retransmissions presumed lost messages. Therefore, TCP is not the best option for real-time applications such as voice over IP communications. However, presently the majority of internet applications, including the World Wide Web (WWW), file transfers, remote administrations, and emails, rely on the TCP transmission.

TCP guarantees that the transmitted packets are received to the destination in the correct order as they were sent. TCP employs acknowledgments to take care of this matter. When a packet is received at the destination, a confirmation will be sent to inform the sender that the packet was received successfully. If an acknowledgment is not received, the sender has to retransmit the packets again after a certain time specified by a specific timer [6]. The retransmission timer is used to ensure data delivery in the absence of any feedback from the remote data receiver. The Retransmission timeout (RTO) is the duration of this time [21].

TCP uses multiple flow and error control mechanisms at the transport level. Inflow control process, a set of segments called the window is transmitted all at once, and each segment is equipped with a sequence number. If the receiver acknowledges the highest part, it is an indication that all of the previous layers have arrived successfully. The sender is informed about the size of the TCP segments window using a field in the TCP header called the advertised window (AWND). This particular information would help the slow receiver not to be overwhelmed by a fast sender. [22]. In the case of a full buffer, some packets might be dropped. Therefore, TCP uses a congestion window (CWND) with a specific size. The sending window is defined as the minimum of the AWND and CWND [6].

TCP connection starts with a three-way handshake protocol. First, the client-side sends a special TCP segment (The SYN segment) to the server-side. The SYN segment contains an SYN bit that equals one and an initial sequence number that is generated randomly by the client-side, for example, SEQ=x. The SYN segment is later encapsulated within an IP datagram and sent to the server. When the server received the IP datagram, it extracts the SYN segment and sends another segment (The SYNACK segment) to the client. The SYNACK segment contains three types of important; the SYN bit, which is set to 1, the acknowledgment field of the TCP segment header that is set to ACK=x+1, and the server chosen initial sequence number SEQ=y. The actual meaning of this segment is, "I received your SYN packet to start a connection with your initial sequence number, x. I agree to establish this connection. My own initial sequence number is y". Once the client receives the SYNACK segment, the client sends another segment to the server with the value y+1 this time in the acknowledgment field, the value x+1 in the sequence number field of the TCP segment header, and the SYN bit is set to zero as the connection is established. Once these three steps completed, the client and server hosts can exchange data. [18]. Figure 1 shows the process of three-ways handshaking in TCP.

B. User Datagram Protocol

User Datagram Protocol (UDP) (RFC 768) is the second widely used protocol in the transport layer. Similar to the TCP, it transfers data across the Internet Protocol IP based network. UDP does not establish a connection between the sender and the receiver. More specifically, UDP does not require a formal handshake to get the data flowing and has no need for SYNs and ACKs flags [23]. Therefore, this protocol is considered unreliable, as there is no guarantee that all of the transmitted data would reach the destination. The messages in UDP are broken into datagrams and sent across the network. Each packet acts as a separate message, and it is handled individually. Datagrams can follow any available path toward the destination, and hence they can be received in a different order [24]. While these were some of the UDP bad marks, yet it is widely used in various types of applications where some data loss is accepted, like the applications that do not require reliable data stream service as multimedia [6]. UDP provides multiplexing at a low level without safe delivery, flow, and congestion control at any network node identified by an IP address.

Additionally, UDP provides checksums for data integrity and to check transmission errors. UDP also contains the port numbers used to address different functions at the source and destination of the datagram. The UDP protocol is extremely simplistic, where Data from the application layer is simply delivered to the transport layer and then encapsulated in a UDP datagram. This datagram is later sent to the host without any mechanism to guarantee a safe arrival to the destination device. If some reliability checking is needed, it is pushed back to the application layer, which may be adequate in many cases. For instance, the IP Multimedia System (IMS) of 3G wireless networks uses the Real-time Transport Protocol (RTP) for exchanging media streams, and RTP typically runs over UDP [25]. UDP-lite (RFC 3828) is a lightweight version of UDP that delivers packets even if their checksum is invalid. This protocol is useful for real-time audio/video encoding applications that can handle single bit errors in the payload [2].

C. Quick UDP Protocol

QUIC is an encrypted, multiplexed, low latency transport protocol that was proposed and implemented by Google in 2012 on top of the UDP and is standardized by the Internet Engineering Task Force (IETF) working group [26]. The UDP is unreliable, and it provides no congestion control. However, this was considered in the QUIC design, which implements congestion control at the application layer [27]. Figure 2 illustrates the difference in the TCP and QUIC mechanisms. QUIC was mainly introduced to improve the HTTPS performance smoothly as is requires no changes to the operating systems [28]. Additionally, QUIC is considered a multiplexed protocol as it multiplexes the streams of different applications in one single connection using a lightweight data structuring abstraction. Along the same side, QUIC, which has UDP as a substrate, is an encrypted transport protocol where packets are authenticated and encrypted. In more detail, QUIC applies a secured cryptographic handshake where known server credentials on repeat connections and redundant handshake at multiple layers are not required any more [29]. Accordingly, handshake latency is limited.

QUIC was proposed by Google to reduce the webpage retrieval time [27]. Therefore, QUIC was deployed in google Chrome [30]. Where the server-side was deployed at Google's servers. At the client-side, QUIC was deployed in three main services; Chrome, YouTube, and the Android Google search app. Improvements were detected at all of the three areas of application where latency was reduced in Google search responses, and the rebuffering time was also minimized in YouTube applications for desktop and mobile users, as presented in Table 1.

Based on such achievements, QUIC accounts now in more than 30% of the total google traffic, which maps to at least 7% of the global internet traffic [31].

Table 1: percentage of reduced latency in google search responses and YouTube application rebuffering after deploying QUIC. (Data from [29].

Google search responses	Desktop users	8%
	Mobile users	3.8%
YouTube playback re-buffer rate	Desktop users	18%
	Mobile users	15.3%

Recently, intermediary devices (Middleboxes) such as firewalls and Network Address Translators (NATs) became vital control points of the Internet's architecture [32]. However, as QUIC implementations incorporate a UDP core, it is not prone to the availability of Middleboxes. Middleboxes might exist along the end-to-end path with a potential effect on the connection-oriented protocols such as TCP, which is not the case with connectionless protocols such as UDP [33]. More specifically, Middleboxes might inspect and modify TCP packet headers. On the other hand, applications that use UDP for communications such as the Domain Name System (DNS) do operate through Middleboxes without employing any additional mechanism.

In client-server models, HTTP/1.1 recommends limiting the single client connections to a server [34], and HTTP/2 recommends using only one single TCP connection between the client and the server [35]. However, it is hard to control the TCP communications framing [36], which consistently causes an additional latency, and such a change might need to modify the webservers, client operating systems, and the mechanisms of the middleboxes to deploy the transport modifications. Additionally, developers and vendors of operating systems and Middleboxes should cooperate. On the contrary, using QUIC built on top of UDP encrypts the transport headers requiring no further change from vendors, developers, or even network operators [29].

QUIC is designed to facilitate a 0-Round Trip Time (RTT) setup of connections using the Diffie-Hellman (DH) exchange. The client initially sends an inchoate message to the server to get a rejection message before the handshake. This rejection message has all server config connection keys required for DH exchanges, such as the server DH public value. Once all server DH keys are available, the client sends

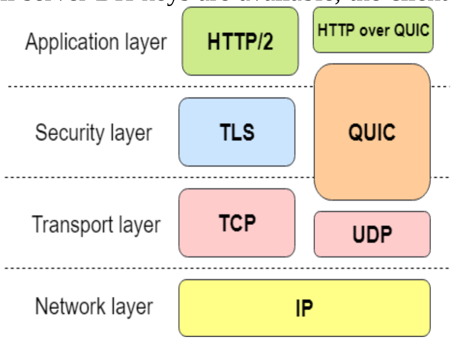

Figure 2: Traditional Quick and TCP stack.

the complete message

containing its public DH value. If the client aims for a 0-RTT, then it must not wait for the server to reply to send the data encrypted with its keys. In other words, if the client has already talked to the server before, the startup latency is 0-RTT, even with encrypted connections [29]. Upon a successful handshake and exchange of the temporary public DH values, the forward-secure keys are calculated at the client and the server, which are directly used to encrypt the exchanged packets. If the server ever changes the config, it replies with the standard rejection message, and the handshake process can start again.

Another issue that QUIC is designed to eliminate the Head-of-Line (HOL) Blocking. The TCP transport is prone to HOL blocking [37] due to sequential delivery resulting from applications multiplexing data in the single-byte stream of TCP's abstraction. However, QUIC multiplies many QUIC streams within the single UDP connection, where many responses or requests can be handled at the same UDP socket [27].

Loss recovery was also handled in QUIC, where the TCP "retransmission ambiguity" problem [38] [39] is fully addressed and avoided. The loss of retransmitted segments can be detected via expensive timeouts. However, in QUIC packets have new packet numbers, including the retransmitted data packets, and thus it is not required to distinguish the retransmission from the original transmission.

QUIC was recently intensively studied, and many recent advances were suggested, such as the Multipath QUIC (MPQUIC), which incorporates a scheduler in order to increase the QUIC throughput and reduce downloading time on the Internet with no required OS changes. In the end, it is noticed that QUIC did not adopt one specific algorithm in all deployments, but it was never reported that QUIC ever generated any harmful traffic to network stability [27].

D. Remote Dynamic Memory Access Technology

The demand on CPU to control the transport of messages The demand for CPU to control the transport of messages through networks has always been a key point that needs further addressing. Many kinds of research have concentrated on minimizing the CPU overhead during packet exchange between network machines. This case was mostly handled by the Remote Dynamic Memory Access technology (RDMA) in which machines are allowed to write/read from/to known memory regions of other network machines without involving their CPUs. With a latency not exceeding (3μs) and zero CPU overhead, RDMA can handle network data exchange [40]. Therefore, in client-server models, the client is allowed to access the server memory to place a service request without involving the server. Besides low latency and low CPU

Overhead, RDMA provides high message rates, which is extremely important to many modern applications that store small objects [41]. The remote memory accesses are accomplished using only the remote Network Interface Card (NIC).

Additionally, Bypassing the kernel in the remote host helped

presenting the RDMA as one of the top available low latency data exchange technologies. RDMA is not widespread as it was long supported by the switch fabric InfiniBand network, which has been expensive with no Ethernet compatibility [42]. However, many resent RDMA advances, such as RoCE [42] [43], are able to work over the Ethernet with data center bridging [44] [45] with reasonable prices. The main difference between the classic Ethernet networks and the RDMA based is the ability of the RDMA NICs to bypass the kernel in all communications and performing hardware-based retransmissions of lost packets [46]. Based on this, the expected latency in RDMA networks is around (1μs), where it is in the classic Ethernet networks around (10μs). Therefore, RDMA is widely used in data centers, especially after the hardware became much cheaper [40].

The availability of the fast networks deploying RDMA technology increased dramatically due to the noticeable drop in their prices. Their prices are now comparable to the costs of the Ethernet [46]. Additionally, the abundance of cheap and large-sized Dynamic Random Access Memories (DRAMs) played a crucial role in spreading the RDMA technology [47]. DRAMs can be used as stores sometime such as Redis [48] or cashes such as Memcached [49]. Internally, they do use many different types of data structures to provide the required memory accesses in the fastest ways.

RDMA uses both transport types; Connected and unconnected. The connected transport methods require a pre-defined connection between the two communicating NICs. Such links might be considered reliable or unreliable based on packet reception acknowledgments. Unreliable connections generate no confirmations, which results in less traffic compared to secure connections. Along the other side, RDMA transport through the unconnected method is implemented through the unreliable datagram, which can perform better only in the one-to-many topologies [46].

In either connection method, RDMA uses seven primary transfer operations (verbs). A Send operation for the standard message transfer to a remote buffer. If the message has a Steering tag (Stag) field, then the remote buffer identified by this Stag will become inaccessible until it receives an enable message. This operation is called send with invalidating. The send with Solicited event is a little different where the remote recipient will generate a specific function when it receives the message. A simple combination is allowed where a message can be invalidating and can carry an event to the remote recipient with the operation Send with Solicited Event and Invalidate. To transfer data, the write operation named RDMA Write is used.

Similarly, to read from a remote buffer, the operation Remote Direct Memory Access Read can be used. The final procedure named Terminate is used to inform the remote recipient about an error that occurred at the sender using a terminate message [50]. Figure 3 a sample RDMA communication.

Employing RDMA in datacenters presented a challenge where RDMA uses a hop by hop flow control and an end to end congestion control [51]. Although congestion control is expected to affect not more than 0.1% of the datacenters employing RDMA [52], multiple flows might collide at a switch causing long latency tails [53] even with good network designs [54]. Many recent projects searched for points that might suffer more in data centers employing the RDMA, and some stated that it occurs only at network edges [52] and suggested using modifications such as Blitz to avoid the p

Blitz uses a divide-and specialize method where the receiver congestion is isolated, which is claimed to speed-up the convergence. Recently, many researchers targeted RDMA technology, which resulted in significant improvements. At the early stages, RDMA could only function in the form of one-sided as any knowledge about the remote process is not needed [40], while the recent advances presented that two-sided RDMA is now possible [55]. Additionally, many new platforms were presented to improve the RDMA performance, such as FaRM, which added two mechanisms to enable efficient use of the single machine transaction in RDMA [42]. Along the same side, HERD [46] is a system that was designed to make the best use of the RDMA networks trying to reduce the RTT using efficient RDMA primitives.

roblem.

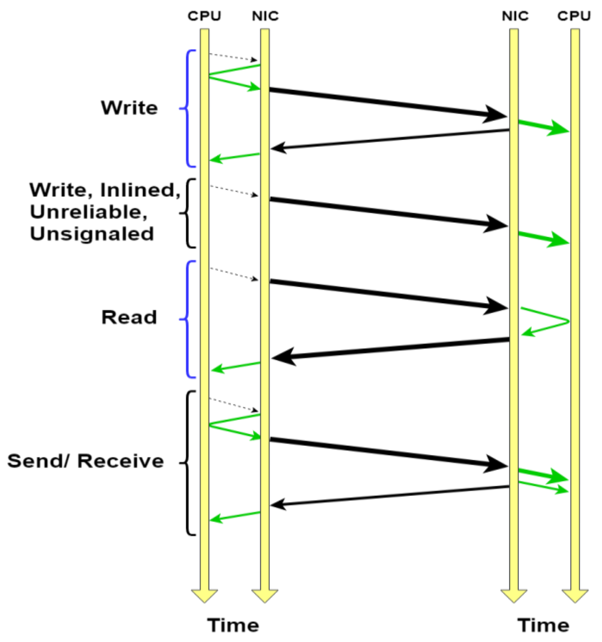

Figure 3: Steps involved in posting verbs. The dotted arrows are PCIe PIO operations. The solid, straight, green arrows are DMA operations: the thin ones are for writing the completion events. The thick solid, straight, black arrows are RDMA data packets and the thin ones are ACKs.

E. Data plane Development Kit

As the number of machines connected to the Internet is increasing day after day, data traffic became an overhead, and more and more CPU cycles are required [56].

Recently, many projects addressed such problems using software solutions instead of relying only on dedicated hardware. Moreover, many researchers suggest that the performance of hardware and software routers is comparable [57]. Therefore, the problems of standard packet processing software need to be all identified at early stages, especially the issue of achieving the 1Gbit/s and 10 Gbit/s network adapters line rate [58]. Many research projects focused initially on creating test models and use different available cases to address these limitations [57] [59]. The results confirmed that the standard software would not be able to reach the maximum performance for small packets. Therefore, many recent projects focused on developing high-speed packet processing frameworks using available and affordable hardware [60] [61] [62]. One last issue that needs to be handled is the network stack, which provides a complete implementation of TCP and UDP protocols in Linux, for example. Full details can be found in [63]. The current form of the network stack is causing multiple problems in memory allocation, which makes achieving the maximum performance almost impossible [58]. The open question remaining, is the current form of network stack suitable or does it need to be extended or even replaced?

At the Intel side, researchers in 2012 presented the Dataplane Development Kit (DPDK) which is a framework that has a set of libraries and drivers that form a lower layer with the required functionalities to process packets in high speed in various data plane applications on intel's architectures [64] [65]. This layer is called the Environment Abstraction Layer (EAL). It can perform all needed tasks such as allocating and managing memory for network packets in addition to buffering the needed packet descriptors in ring-like structures then pass them through the NIC to the destination application (and vice versa) [56].

In more details, the packet processing is completely done at the user-space, which requires Huge-pages for storage. The DPDK supports 32-bit and 64-bit intel possessors from Atom to Xeon generations with or without non-uniform memory access (NUMA) regardless of the number of cores or processors available.

DPDK runs on the Linux operating system with full replacement to the network stack. It is implementing the so-called "run-to-completions" model where all resources must be allocated before the data plane applications running on the processor cores. Congestion control is not widely questioned in DPDK, while scheduling is not supported, as all devices must be accessed via polling [58]. Additionally, work can be done in stages as cores can exchange messages.

DPDK LIBRARIES

To make packet processing faster, DPDK provides various libraries that were optimized for high performance. These libraries are executed in user space and perform all necessary tasks similar to the Linux network stack, such as buffering packet descriptors, allocating memory for packets, and passing the packets to the application (and vice versa) through the NIC. In simple words, the libraries manage the memory, queues, and the buffers.

1. *Queue Management*

The provided library manages any type of queues by delivering a ring structure with a fixed size with doubly linked list implementation following the FIFO principle with en-queue and de-queue standard functionalities [13]. Although the lockless application ensures a faster writing process [66], the size of the structure is fixed and not resizable during the runtime.

2. *Memory Management*

The mapping to the physical memory is handled by the EAL [12], where every object and structure is granted a specific portion of the physical memory. For maximum performance, suitable padding is added between objects to ensure that they are equally loaded [12]. The only concern is that multiple cores might be needed to access the ring, which might result in a bottleneck unless each core has a separate local cash memory [58]

3. *Buffer Management*

DPDK has a designed library to transport network packets. This library provides the needed buffers which are created before any application's runtime. During the runtime, the user must be able to pick and use any available buffer and must free it after wise. The size of the buffers is normally small, but in case larger packets are transferred, multiple sets of them are chained as it has pointers to another one. Uses a library that provides the ability to allocate and free buffers used by DPDK applications to store message buffers. It reduces the time the operating system spends in allocating and deallocating buffers. DPDK pre-allocates fixed-size buffers.

The DPDK has many other side libraries performing different tasks like the Longest Prefix Matching (LPM) library, which helps to implement a table for an algorithm to forward packets based on the IPv4 address [67]. Additionally, DPDK also has a hash library that is used for fast lookups of a specific unique key in a big set of entries [12].

DPDK Operation

Once the EAL is created, the DPDk provides all libraries as the EAL must have access to all low-level resources such as the hardware and the memory to create the needed interfaces [65]. At this stage, all applications can be launched and loaded, all memory space processes can be accomplished, and all high-speed packet-processing functionalities are ready. However, DPDK provides no features to handle the firewalls or TCP/UDP as in the standard Linux network stack and relays of the programmers to build them.

Recently, many research groups have focused on enhancing the DPDK functionalities, and many new versions are offered, such as the DPDK v Switch [68], HPS Router [8], and Open Function [69].

IV. THOROUGH COMPARISON AND EVALUATION

In this section, we present a detailed comparison between the studied data paths according to the techniques, latency and congestion control, the achieved throughput, middle boxes consideration, loss recovery mechanisms, developer productivity, and host resource utilization.

A. Techniques

The packet transfer process is accomplished using different methods in different data paths. TCP, for example, is a connection-oriented point to point protocol where a process can start sending data to another after a "handshake" where a set of preliminary segments are exchanges to establish the parameters of the ensuing data transfer [18]. TCP ensures that all sent packets are received and in the correct order and thus considered a reliable technique.

However, UDP is considered a connectionless and unreliable protocol as it does not require a formal handshake to initiate the data flow. Although UDP provides a basic data multiplexing, another protocol using the unreliable techniques is the QUIC which is a multiplexed encrypted protocol implemented by Google on top of the unreliable UDP to improve the performance of the HTTPS without any change to the operating system. The encryption and decryption are accomplished through a kind of cryptographic handshake where known server credentials on repeat connections are used. On the other side, different mechanisms are applied in the RDMA and DPDK. Using RDMA machines are allowed to write/read from/to known memory regions of other network machines without involving their CPUs. In DPDK, a set of libraries and drivers form a lower layer with the needed functionalities to process packets at high speed in data plane applications on intel’s architectures.

B. Latency and Congestion Control

Latencies and losses frequently occur because of network congestions, and thus packet drops are induced. Usually, the congestion control uses "Congestion Window" to determine the number of bytes a network can handle. However, the size of this has no standard [70].

The TCP protocol standards remain unaware of the unexpected effects that happened at network resources on the Internet, including the appearance of congestion collapse [71]. In a more detailed description, TCP requires 1 RTT for the handshake and 1 or 2 in case of encrypted communication. At the same time, other protocols like QUIC would need at most 1 RTT for first-time communication and 0 RTT in case there was a communication previously between this host and server [27]. The reason behind this difference is that QUIC deploys intensive multiplexing, where streams of different applications are multiplexed in one single connection using a lightweight data structuring abstraction. RDMA presents another high-speed communication where eliminating kernel/user context switches, and it provides low latency, not exceeding (3μs) and zero CPU overhead [40]. Besides, RDMA provides many congestion control mechanisms [19]. DPDK had a different way of dealing with latency, where developers using DPDK in applications proved that it could reduce the latency up to 2 times compared to other available technologies [72]. However, the user of UDP knows that it implements no congestion control [27].

C. Head of Line Blocking

It is a common problem in computer networks where the line of packets is held up by the first packet for various reasons. In TCP, it is frequently caused by out-of-order packet delivery [27]. For example, HTTP/2 still suffers from HOL using TCP, where one lost packet in the TCP stream makes all streams wait until that package is retransmitted and received. However, in UDP, if one packet is lost, it doesn't stop the data from getting delivered as UDP doesn't force the order. So, there is no HOL in UDP neither in the QUIC protocol implemented on top of it. In RDMA, it was a different case wherein early stages HOL did not get much attention as it is considered really rare to occur, but modern RDMA frameworks such as DCQCN [73] pointed to this problem and solved through introducing an end-to-end congestion control scheme.

D. Throughput

A network throughput refers to the amount of data moved successfully from one place to another in a given time period. It normally depends on the congestion window or the buffer size estimated by the congestion controller at the sender's side and the receiving window at the receiver. At TCP, the efficiency is the most important factor to tune the parameters, and it is noticed that default parameters of TCP are designed to sacrifice throughput, in exchange for an efficient sharing of bandwidth on congested networks [74]. Regarding the buffer size, various techniques tried to determine the optimal size to achieve the best performance [75]. It is reported that in TCP to make the best throughput, the buffer size should be dynamically adjusted to the connection and server characteristics through dynamic right-sizing [76] and buffer size allocation [77]. On the other hand, the UDP measured throughput is better than TCP under the same conditions. The reason behind this is that UDP does not require acknowledgments, but the downside is that sometimes small fragments can be lost [74]. In the modern data paths such as QUIC, an advertised connection flow limit is specified by 15 MB [29], which is considered large enough to avoid flow control bottlenecks. Compared to TCP, for example, in video transfer, QUIC increased the number of videos that can be played at their optimal rates by 20.9% [29]. The highest throughput to be achieved is using the

RDMA were offloading the networking stack, and eliminating kernel/user context switches resulted in low latency and high throughput. In Datacenters, servers would achieve the highest throughput using RDMA [46]. In DPDK's early stages, the performance was not on the top of the pyramid as the bottleneck was the CPU with a limited number of cycles per second [58]. However, as the door is open to developers to help to improve the performance, many new implementations focused on increasing the throughput up to 13x times [78].

E. Middleboxes

Middleboxes are network appliances that are capable of manipulating, filtering, inspecting, or transforming the traffic, which might affect the packet forwarding. The way to deal with middleboxes differentiates old and new data paths. TCP and UDP are long working protocols even before most of the currently available middleboxes. Therefore, middleboxes are not affecting the standard TCP, or UDP traffics. TCP, for example, remains a trusted backup for new data paths so that if their transmission is blocked through a middlebox, the transfer is switched to TCP to ensure successful communication [29]. QUICK packets were all blocked by most middleboxes in the early stages. Later, a vast range of middleboxes was identified, and the vendors were reached out, and presently QUIC traffic is treated reasonably in most middleboxes [29]. In RDMA, the middleboxes are overlooked as datacenters usually don't have middleboxes [79]. However, DPDK went further and created unique versions that are applicable for middleboxes such as OpenFunction [69].

F. Loss Recovery and Retransmission Ambiguity

In TCP, the "retransmission ambiguity" problem was raised as frequently the receiver could not figure whether the received acknowledgment was for an original transmission (presumed lost) or for a retransmission. The loss of a retransmitted segment in TCP is commonly detected via an expensive timeout [29]. However, each QUIC packet carries a unique packet number, including those taking retransmitted data. Therefore, no further mechanisms are needed to distinguish a retransmission from an original transmission, thus avoiding TCP's retransmission ambiguity problem. Other technologies such as RDMA use lossless link-level flow control, namely, credit-based flow control and Priority Flow Control, where packets are almost never lost even with unreliable transports. However, Hardware failures might lead to losing some packets, which are extremely rare [46]. DPDK also had its own techniques where the dropout rate was studied in detail in terms of packet size, hash tables, and the packet buffering memory (membool). It was found the small packet sizes, complex hash tables, and extremely small or extremely large membools are associated with more dropped packets [80]. On the other hand, UDP employs no loss recovery techniques.

G. Developer productivity

Data paths provide the developers with multiple developments or testing tools to safely use the data path. For example, in serverless communications, the Lambda model [81] can be used by developers to check responses to events such as in Remote Procedure Calls (RPC), mostly based on TCP connections. On the browsing side, SIP APIs help the developers manage, test and analyze the TCP/UDP transport resulted from browsing. Similarly, Silverlight and Flash player are excellent developer tools, but it is noticed that TCP is preferred over UDP therein [82]. However, QUIC testing and analysis are provided through the developer tools of the Chrome browser, which helps to measure the time elapsed since requesting a web page until the page is fully loaded. In RDMA and DPDK, it was more flexible where recent RDMA frameworks are customizable, and developers can assign algorithms and even design the storage structure as in Derecho [83]. Similarly, the DPDK frameworks allow the developers to write their network protocol stacks and implement a set of high-performance datagrams forwarding routines in the user space [8].

H. Host resource utilization

In TCP/UDP, a defined memory should be reserved at both communication sides, and the used data structure must be shared across the different CPU threads/processes, especially in multi-core CPU architectures [84]. The same applies to ports, where an allocated port for the communication will be allocated for the whole conversation. Similarly, the QUIC implementation was initially written with a focus on rapid feature development and ease of debugging, not the CPU efficiency. Even after applying several new mechanisms to reduce the CPU cost, QUIC remained more costly than TCP [29]. However, RDMA reduces CPU load, as it bypasses the kernel and the TCP/IP stack and avoids data copy between the user space and the kernel [43]. However, it still needs special access to the destination memory. At the DPDK side, the bottleneck is the CPU. A CPU can only operate a limited number of cycles per second. The more complex the processing of a packet is, the more CPU-cycles are consumed, which then cannot be used for other packets. This limits the number of packets processed per second. Therefore, in order to achieve higher throughput, the goal must be to reduce the per-packet CPU-cycles [58].

V. CONCLUSION

Improvements to the traditional protocols are possible but not urgently required. QUIC, which was built on top of UDP, outperformed UDP in many aspects. Compared to TCP, the RDMA throughput is higher, the QUIC is faster and better for transferring videos in their optimal rate, and DPDK works better in Intel architectures. However, the long-term supported protocols TCP and UDP are part of many standard communications and applications, and probably they will persistently play a key role in future data transmissions. This work showed it is useful to direct the protocol used to some specific areas where they fit the best. RDMA provides in the communication of data centers, QUIC would help developers in multi-disciplines due to the available routines connecting with the application layer, and DPDK would work correctly in appliances with Intel architecture.

VI. REFERENCES

[1] M. A. H. Inam Ullah Khan, "Transport Layer Protocols And Services," *International Journal of Research in Computer and Communication Technology,* pp. 2320-5156, 2016.

[2] J. H. Mehrnoush Rahmani, "A comparative study of network transport protocols for in-vehicle media streaming," *IEEE International Conference on Multimedia and Expo,* pp. 441-444, 2008.

[3] D. W. Tanenbaum Andrew, "Computer networks, fourth edition," *Prentice hall,* 2003.

[4] B. F. Zafar Saima, "A survey of transport layer protocols for wireless sensor networks," *International Journal of Computer Applications*, 2011.

[5] S. William, "Business data communications," *Prentice Hall PTR,* 1990.

[6] A. K. Purnya Awasthi, "Comparative Study and Simulation of TCP and UDP Traffic over Hybrid Network with Mobile IP," *International Journal of Computer Applications*, 2013.

[7] G. Bhargavi, "Experimental Based Performance Testing of Different TCP Protocol Variants in comparison of RCP+ over Hybrid Network Scenario," *International Journal of Innovations & Advancement in Computer Science IJIACS ISSN,* pp. 2347-8616, 2014.

[8] Zongyao Li, "HPSRouter: A high performance software router based on DPDK," in *2018 20th International Conference*, 2018.

[9] Rizzo Luigi, "Netmap: a novel framework for fast packet I/O," *21st USENIX Security Symposium (USENIX Security 12),* pp. 101-112, 2012.

[10] M. C. M. B. Wenji Wu, "The performance analysis of Linux networking–packet receiving," *Computer Communications*, pp. 1044-1057, 2007.

[11] R. B. Raffaele Bolla, "Linux software router: Data plane optimization and performance evaluation," *Journal of Networks,* pp. 6-17, 2007.

[12] I. DPDK, "Data Plane Development Kit Project," *http://www.dpdk.org,* 2014.

[13] I. DPDK, "Programmers Guide," 2014. [Online].

[14] N. T. Solutions, "ATM Traffic Management White paper last accessed," [Online]. [Accessed 3 January 2019].

[15] H. Al-Bahadili, "Simulation in computer network design and modeling: Use and analysis," *Hershey, PA: IGI Globa,* p. 282, 2012.

[16] H. M. M. E. Z. S. G. Liu Hang, "Error control schemes for networks: An overview," *Mobile networks and Applications,* pp. 167-182.

[17] X. Zuo, M. Wang, T. Xiao and X. Wang, "Low-Latency Networking: Architecture, Techniques, and Opportunities," *IEEE Internet Computing 22*, 2018.

[18] James Kurose, Computer Networking A top down approach, 2013.

[19] KL Dias, JAS Monteiro, D Florissi , "Traffic Management in Isochronets Networks," *Managing QoS in Multimedia Networks and Services,* pp. 131-146, 2000.

[20] W. Richard, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms.," *Network Working Group, Request for Comments: 2001,* 1997.

[21] S. Vern Paxson, "Computing TCP's retransmission timer," *RFC 6298,* 2011.

[22] P. Rajashree, "Selective-TCP for wired/wireless networks," *School of Computing Science-Simon Fraser University,* 2006.

[23] M. Rasheed, NM Norwawi, O Ghazali,"Detection algorithm for internet worms scanning that used user datagram protocol," *International Journal of Information and Computer Security,* pp. 17-32, 2019.

[24] A. Bonczkowski Joshua, "Computer implemented system and method and computer program product for testing a software component by simulating a computing component using captured network packet information," *U.S. Patent 9,916,225*, 2018.

[25] S. Addagatla, B Goddanakoppalu, "System and method of network congestion control by UDP source throttling," *U.S. Patent Application 10/758,854*, 2005.

[26] "IETF QUIC working group.," [Online].

[27] G Carlucci, L De Cicco, S Mascolo, "HTTP over UDP: an Experimental Investigation of QUIC," *Proceedings of the 30th Annual ACM Symposium on Applied Computing,* pp. 609-614, 2015.

[28] S. Tobias Viernickel, "Multipath quic: A deployable multipath transport protocol," *IEEE International Conference on Communications*, pp. 1-7, 2018.

[29] ARAWAVAVCKDZFYF, B. Adam Langley, "The QUIC transport protocol: Design and Internet-scale deployment," *Proceedings of the Conference of the ACM Special Interest Group on Data Communication,* pp. 183-196, 2017.

[30] "QUIC, a multiplexed stream transport over UDP," [Online].

[31] Sandvine, "Global Internet Phenomena Report - Latin America and North America," 2016.

[32] U. Shan Huang, "Middleboxes in the Internet: a HTTP perspective," *Network Traffic Measurement and Analysis Conference (TMA),* pp. 1-9, 2017.

[33] S. Eggert, "UDP usage guidelines," *RFC 8085,* 2017.

[34] J. R. Fielding Roy, "RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," *Internet Engineering Task Force (IETF),* 2014.

[35] R. P. a. M. T. Belshe Mike, "RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2)," *Internet Engineering Task Force (IETF),* 2015.

[36] D. L. T. Clark David D., "Architectural considerations for a new generation of protocols," *ACM SIGCOMM Computer Communication Review,* pp. 200-208, 1990.

[37] S. K. Scharf Michael, "Head-of-line Blocking in TCP and SCTP: Analysis and Measurements," *GLOBECOM,* pp. 1-5, 2006.

[38] C. P. Karn Phil, "Improving round-trip time estimates in reliable transport protocols," *ACM SIGCOMM Computer Communication Review,* pp. 2-7, 1987.

[39] Z. Lixia, "Why TCP timers don't work well," *ACM SIGCOMM Computer Communication Review,* pp. 397-405, 1986.

[40] L. Mitchell Christopher, "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store," *USENIX Annual Technical Conference,* pp. 103-114, 2013.

[41] P. Atikoglu Berk, "Workload analysis of a large-scale key-value store," *ACM SIGMETRICS Performance Evaluation Review,* pp. 53-64, 2012.

[42] H. Dragojević Aleksandar, "FaRM: Fast remote memory," *{USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14),* pp. 401-414, 2014.

[43] B. Shpiner Alexander, "Roce rocks without pfc: Detailed evaluation," *Proceedings of the Workshop on Kernel-Bypass Networks,* pp. 25-30, 2017.

[44] IEEE, "802.1Qau - Congestion Notification," 2010. [Online].

[45] IEEE, "802.1Qbb - Priority-based Flow Control," 2011. [Online].

[46] A. Kalia Anuj, "Using RDMA efficiently for key-value services," *ACM SIGCOMM Computer Communication Review,* pp. 295-306, 2015.
[47] C. Dragojevic Aleksandar, "RDMA Reads: To Use or Not to Use?," *IEEE Data Eng. Bull,* pp. 3-14, 2017.
[48] "Redis: An Advanced Key-Value Store," [Online].
[49] "memcached: A Distributed Memory Object Caching System," 2011. [Online].
[50] G. Recio Renato, "A Remote Direct Memory Access Protocol Specification," *RFC 5040,* 2007.
[51] S. Gran Ernst Gunnar, "First experiences with congestion control in InfiniBand hardware," *arallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium* , pp. 1-12, 2010.
[52] T. Xue Jaichen, "Fast Congestion Control in RDMA-based Datacenter Networks," *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos,* pp. 24-26, 2018.
[53] S. Alizadeh Mohammad, "Data center tcp (dctcp)," *ACM SIGCOMM computer communication review,* pp. 63-74, 2011.
[54] L. Charles, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers 100,* pp. 892-901, 1985.
[55] A. Kalia, "FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs," *OSDI,* pp. 185-201, 2016.
[56] S. A. R. P. P. A. N. P. Hrishikesh Kulkarni, "A survey on TCP/IP API stacks based on DPDK," *Internation Journal Of Advance Research And Innovative Ideas In Education* , 2017.
[57] R. Dobrescu Mihai, "RouteBricks: Exploiting Parallelism To Scale Software Routers," *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles,* pp. 15-28, 2009.
[58] S. Dominik, "A look at Intel's dataplane development kit," *Network*, 2014.
[59] R. B. Bolla Raffaele, "Linux software router: Data plane optimization and performance evaluation," *Journal of Networks,* pp. 6-17, 2007.
[60] M. Han Sangjin, "PacketShader: a GPU-accelerated software router," *ACM SIGCOMM Computer Communication Review,* pp. 195-206, 2011.
[61] L. D. Fusco Francesco, " High speed network traffic analysis with commodity multi-core systems," *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement,* pp. 218-224, 2010.
[62] P. Bonelli Nicola, "On multi–gigabit packet capturing with multi–core commodity hardware," *International Conference on Passive and Active Network Measurement,* pp. 64-73, 2012.
[63] L. Rio Miguel, " A map of the networking code in Linux kernel 2.4. 20," *Technical Report DataTAG-2004–1,* 2004.
[64] I. DPDK, "Packet Processing on Intel Architecture, Presentation slides," 2012. [Online].
[65] I. DPDK, "Getting Started Guide," 2014. [Online].
[66] l. A. l. ring-buffer, "http://lwn.net/Articles/340400/," 2014.
[67] B. Christian, "Understanding Linux network internals," *O'Reilly Media, Inc,* 2006.
[68] I. O. S. T. C. P. Processing, " https://01.org/packet-processing," 2014. [Online].
[69] Z. Tian Chen, "OpenFunction: Data Plane Abstraction for Software-Defined Middleboxes," *arXiv preprint arXiv:1603.05353,* 2016.
[70] E. H. T. A. Badach, "echnik der IP-Netze -TCP/IP incl. IPv6," *Funktionsweise, Protokolle und Dienste.*
[71] K. Afanasyev Alexander, "Host-to-host congestion control for TCP," *IEEE Communications surveys & tutorials,* pp. 304-342, 2010.
[72] H. Mao Chen-Nien, "Minimizing latency of real-time container cloud for software radio access networks," *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference,* pp. 611-616, 2015.
[73] Z. Zhu Yibo, "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM Computer Communication Review,* pp. 523-536, 2015.
[74] K. Giannoulis S., "TCP vs. UDP Performance Evaluation for CBR Traffic On Wireless Multihop Networks," *Simulation 14*, 2009.
[75] T. Brian, "TCP Tuning Guide for Distributed Applications on Wide Area Networks," *Usenix & SAGE Login,* pp. 33-39, 2001.
[76] F. Weigle Eric, "Dynamic right-sizing: A simulation study," *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference,* pp. 152-158, 2001.
[77] R. C. Cohen Amit, "A dynamic approach for efficient TCP buffer allocation," *IEEE Transactions on Computers,* pp. 303-312, 2002.
[78] V. DeCandia Giuseppe, "Scalable Key/Value Search in Datacenters," *ACM SIGOPS operating systems review,* pp. 205-220, 2007.
[79] R. Gandhi, "improving cloud middlebox infrastructure Online Services," *Purdue University, Theses and Dissertations,* 2016.
[80] L. Xiaoban Wu, "Network Measurement for 100Gbps Links Using Multicore Processors," *The 3rd Innovating the Network for Data-Intensive Science,* pp. 1-10, 2016.
[81] D. Hendrickson Scott, "Serverless computation with openlambda," *Elastic,* p. 80.
[82] W. Davids Carol, "SIP APIs for voice and video communications on the web," *Proceedings of the 5th International conference on principles, systems and applications of IP telecommunications,* 2011.
[83] R. Birman Ken, "Groups, Subgroups and Auto-Sharding in Derecho: A Customizable RDMA Framework for Highly Available Cloud Services," *NSDI '17,* 2016.
[84] A. K. P. Woo Shinae, "Scalable TCP session monitoring with symmetric receive-side scaling," *KAIST, Daejeon, Korea, Tech. Rep,* 2012.