



Traditional vs. Modern Data Paths: A Comprehensive Survey

Ahmad Barghash ^{1,*}, Lina Hammad ² and Ammar Gharaibeh ³

¹ Department of Computer Science, German Jordanian University, Madaba 11180, Jordan

² Data Science and Artificial Intelligence Department, Al Hussein Technical University, Amman 91929, Jordan

³ Department of Computer Engineering, German Jordanian University, Madaba 11180, Jordan

* Correspondence: ahmad.barghash@gju.edu.jo

Abstract: Recently, many new network paths have been introduced while old paths are still in use. The trade-offs remain vague and should be further addressed. Since last decade, the Internet is playing a major role in people's lives, and the demand on the Internet in all fields has increased rapidly. In order 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 in order 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 Quick UDP Internet Connections (QUIC), remote direct memory access (RDMA), and the Data Plane Development Kit (DPDK). This raised the question on which data path should be adopted and based on which features. In this work, we try to answer this question using different perspectives such as the protocol techniques, latency and congestion control, head of line blocking, the achieved throughput, middleboxes consideration, loss recovery mechanisms, developer productivity, host resources utilization and targeted application.

Keywords: data path; TCP; UDP; QUIC; RDMA; DPDK



Citation: Barghash, A.; Hammad, L.; Gharaibeh, A. Traditional vs. Modern Data Paths: A Comprehensive Survey. *Computers* **2022**, *11*, 132. <https://doi.org/10.3390/computers11090132>

Academic Editor: Josip Lorincz

Received: 14 July 2022

Accepted: 17 August 2022

Published: 31 August 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The availability of electronic appliances that store various types of data or have the ability to hold desired services lead to the demand of sharing data or services with other electronic appliances. Computer networks, for example, facilitate the sharing of different types of data between computers, laptops, or any smart devices that can connect to the network. Network services can also be shared through a network such as a printer or a 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 last decade, the largest known computer network in the world is the Internet. The Internet plays a key role 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 five logical layers in order 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 transport layer. Related process from other layers will be discussed as needed.

The transport layer tries to ensure packets are error-free prior to 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 process-to-process delivery. The application data at the transport layer appear in the forms of segments that

are ready for transmission, and all segments 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. Connection-oriented protocols start by establishing a virtual connection before transferring segments from the source to the destination and terminates the connection once all data are transferred. Such protocols normally use certain mechanisms to control the flow and transmission errors. On the other hand, connectionless protocols do not establish connections prior to sending the data [2], and every single segment is treated as an independent packet [3–5]. 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. The 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 the message delivery. If any error occurs during transmission, the packet will be automatically resent over the network [7]. On the other hand, the UDP employs a simpler transport model with a minimum of protocol technique. Additionally, using the UDP, it is possible to transmit messages indicated as datagrams with various Internet services such as emails, webpages, and video streaming.

At the hardware side, network appliances based on dedicated middleboxes are also essential to the network system and have a major effect on performance and reliability. The main task of the appliances is to forward and filter packets, but 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 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 different vendors of the appliances. Third, the types of the middleboxes are increasing daily, which makes it more difficult for administrator to master their new operating systems. Additionally, hardware-based network appliances are space consuming and have poor flexibility. Finally, their development is slow, and it will take years to catch up to the development of new network technologies [8].

In 2012, many large carriers proposed the concept of network function virtualization (NFV) [9], in order 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 an easy progress. The huge obstacle was the gap in the software performance compared to the hardware performance. Although Unix-based software routers provide high flexibility and a low cost, the packet processing is slow, and the huge delay is unacceptable for network systems. Therefore, many projects concluded that standard software is not suitable for high-speed packet-forwarding scenarios [9–11]. However, various research groups have analyzed the bottleneck of network performance by building models to leverage the available network data paths or build better ones. A network data path describes the set of functional units that perform the packet processing operations.

To this end, Google proposed the Quick UDP Internet Connections (QUIC) Protocol, which is an encrypted and multiplexed transport protocol. Another protocol deploys remote dynamic memory access (RDMA) technology, which allows one to write/read data to/from known memory regions of other network machines without involving their operating systems. The last protocol involved in this study was proposed by Intel Company. It is a high-speed packet-processing framework named Data plane Development Kit (DPDK) [12,13]. Here, a lower layer is implemented that performs all tasks such as

allocating and managing memory for network packets, in addition to buffering the needed packet descriptors and passing them through the network interface card (NIC).

In this work, we try to answer the question of which data path should be adopted 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 terms used throughout the paper. Section 3 presents an overview about network data paths. A comparison between the studied network data paths is presented in Section 4. A set of performed simulation results is presented in Section 5. Finally, we conclude the paper in Section 5.

2. Network Data Path

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

2.1. The Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) (RFC 0793) is one of the most famous protocols in the Internet protocol suite and is considered the cornerstone protocol in the transport layer [1]. The 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.

The 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 in order to establish the parameters of the ensuing data transfer [14]. The TCP service is considered reliable, ordered, and error-checked. The TCP was optimized for accurate delivery not the timely delivery. Long delays might occur waiting for out-of-order messages or re-transmissions of presumed lost messages. Therefore, the 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. Moreover, TCP is frequently labeled as “Old But Gold” [15], where it is consistently proposed to assist in modern network demands including examining the DNS latency, as it has recently been reported that DNS TCP connections are a useful source of latency information about anycast services for DNS based on tests sending European traffic to Australia.

The TCP aims to guarantee that the transmitted packets are received to the destination in the correct order as they were sent. The TCP employs acknowledgements, sequence numbers, and timers to take care of this matter. When a packet is received at the destination, an acknowledgement is sent to inform the sender that the packet was received successfully. If an acknowledgement 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 timer [16].

The performance of TCP depends on the used congestion control algorithm. However, traditional algorithms face many challenges and limitations in modern applications. Therefore, congestion control techniques are subject to receive further enhancements including the usage of machine learning techniques, as proposed in improving TCP congestion control in 5G networks [17].

2.2. User Datagram Protocol (UDP)

The 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.

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 the 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, 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 [18]. Moreover, UDP is one of the traditional data paths, but it is consistently discussed to solve problems in modern networks such as its proposed role narrowband internet of things (NB-IoT) communications [19] or the role of UDP-Lite for standard IoT networks [20].

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]. However, UDP does not employ congestion control mechanisms by itself, and applications using UDP should choose suitable mechanisms to prevent congestion collapse [21]. Therefore, many techniques were built on top of the UDP to for more reliability and congestion control [22,23].

2.3. Quick UDP Internet Connection (QUIC) Protocol

Quick UDP Internet Connection 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 [24]. 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 [25].

QUIC was mainly introduced to improve the HTTPS performance smoothly, as it requires no changes to the operating systems [26]. 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. Furthermore, QUIC, which has UDP as a substrate, is an encrypted transport protocol where packets are authenticated and encrypted. In more details, QUIC applies a secured cryptographic handshake, where known server credentials on repeated connections and redundant handshake at multiple layers are not required anymore [27]. Accordingly, handshake latency is limited.

QUIC was proposed by Google to reduce the webpage retrieval time [25]. Therefore, QUIC was deployed in Google Chrome [28], where the server side was deployed in 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 three areas of application. Latency was reduced in Google search responses, and the rebuffering time was minimized in YouTube applications for desktop and mobile users, as presented in Table 1.

Table 1. Percentage of reduced latency in Google search responses and YouTube application rebuffering after deploying QUIC [27].

Latency Type	Users	Percentage
Latency reduction in Google search responses	Desktop users	8%
	Mobile users	3.8%
YouTube playback re-buffer rate reduction	Desktop users	18%
	Mobile users	15.3%

Based on such achievements, QUIC now accounts for more than 30% of total Google traffic, which maps to at least 7% of global Internet traffic [29].

Recently, intermediary devices (middleboxes) such as firewalls and network address translators (NATs) became key control points of the Internet's architecture [30]. However, as QUIC implementations incorporates 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, while this is not the case with connectionless protocols such as UDP [21]. 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 mechanisms.

In client server models, HTTP/1.1 recommends limiting the single client connections to a server [31], and HTTP/2 recommends using only one single TCP connection between the client and the server [32]. However, it is hard to control the TCP communications framing [33], which consistently causes an additional latency, and such a change might require modifications to the web servers, client operating systems, and the mechanisms of the middleboxes. Additionally, developers and vendors of operating systems and middleboxes should cooperate. On the contrary, deploying QUIC built on top of UDP encrypts the transport headers, requiring no further changes from vendors, developers, or even network operators [27].

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 in the Internet without changes to the operating systems. At 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 the network stability [25].

2.4. Remote Dynamic Memory Access (RDMA) Technology

The demand on CPU to control the transport of messages through networks has always been a key point that needs further addressing. Many studies have concentrated on minimizing the CPU overhead during packet exchange between network machines. This case has mostly been handled by remote dynamic memory access (RDMA) technology, in which machines are allowed to write/read data to/from known memory regions of other network machines without involving their CPUs. With a latency not exceeding 3 μ s and zero CPU overhead, RDMA can handle network data exchange [34]. 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 are extremely important to many modern applications that store small objects [35]. The remote memory accesses are accomplished using only the remote network interface card (NIC).

Recently, many researchers have targeted the RDMA technology, which resulted in major improvements. In its early stages, RDMA could only function in a one-sided form, where knowledge about the remote process is not required [34], while recent advances have made two-sided RDMA possible [36]. Additionally, many new platforms have been presented to improve the RDMA performance, such as fast remote memory (FaRM) [37], which added two mechanisms to enable an efficient use of single machine transaction in RDMA. Similarly, HERD [38] is a system that was designed to make the best use of the RDMA networks trying to reduce the RTT using efficient RDMA primitives.

Employing RDMA in datacenters presented a challenge where RDMA uses a hop-by-hop flow control and end-to-end congestion control [39]. Although congestion control is expected to affect not more than 0.1% of the datacenters employing RDMA [40], multiple flows might collide at a switch, causing long latency tails [41] even with good network designs [42]. Many projects suggested that the critical points in datacenters employing RDMA exist at network edges [41], and several modifications have been suggested to handle this issue, such as the Blitz proposal [40].

2.5. Data Plane Development Kit (DPDK)

As the number of machines connected to the Internet is increasing day after day, data traffic has become an overhead, and more CPU cycles are required [43]. Recently, several projects have addressed such problems using software solutions instead of relying only on dedicated hardware. Moreover, it is suggested that the performance of hardware and software routers is comparable [44]. Therefore, all the problems of common packet

processing software need to be identified at early stages, especially the issue of achieving 1 Gbit/s and 10 Gbit/s network adapter line rates [45].

To address these possible limitations, many research projects have initially focused on creating test models and addressing different cases [11,44]. The results have confirmed that standard software will not be able to reach maximum performance for small packets. Therefore, it is necessary to develop high-speed packet-processing frameworks using available and affordable hardware [46–48]. 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, which has been fully addressed by Rio et al. [49]. The current form of the network stack is causing multiple problems in memory allocation, which makes achieving maximum performance almost impossible [45]. The question remaining is if the current form of network stack is suitable, or if it needs to be extended or even replaced.

Intel researchers proposed in 2012 a solution through the Data Plane Development Kit (DPDK). It is a framework that has a set of libraries and drivers forming a lower layer with the required functionalities to process packets in high speed in various data plane applications on Intel’s architectures [50,51]. This layer is called the environment abstraction layer (EAL) and 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) [43]. In more detail, packet processing is entirely performed at the user space, which requires a large number of pages to store. The DPDK supports 32-bit and 64-bit Intel processors from Atom to Xeon generations with or without non-uniform memory access (NUMA), regardless of the number of cores or processors available.

Recently, many research groups have focused on enhancing DPDK functionalities, and many new versions are offered, such as the DPDK vSwitch [52], HPSRouter [8], and OpenFunction [53]. One of the major enhancements is to build a single congestion control algorithm on top of the DPDK (such as NewReno), as it is a Kernel bypass library, and an extra algorithm for congestion control will be required.

3. Comparison of Data Paths

In this section, we present a detailed comparison between the studied data paths according to the techniques, latency and congestion control, head of line blocking, the achieved throughput, middleboxes consideration, loss recovery mechanisms, developer productivity, host resources utilization, and target applications. At the end, we present a summary of the comparison in Table 2.

Table 2. A comparison summary of the chosen data paths.

	Reliability	Congestion Control	HOL	Throughput	Middleboxes Effect	Recovery of Lost Packets	Example of Target Applications
TCP	Reliable	Available	Prone to	Not Important	Low	Recovered	Emails and Web browsing
UDP	Unreliable	Not Available	No HOL	High	Low	Not recovered	VoIP
QUIC	Contains unreliable component	At the application layer	No HOL	High	High	Recovered	Streaming
RDMA	Reliable	Available	Prone to	Top	Low	Lossless links	Datacenters
DPDK	Unreliable	Not Available	Not reported	High	Low	-	Intel processors

3.1. Techniques

3.1.1. TCP

TCP is a connection-oriented point-to-point protocol, where a process can start sending data to another after a “handshake”. In handshaking, a set of preliminary segments are exchanged to establish the parameters of the ensuing data transfer [14]. TCP ensures

that all sent packets are received and in the correct order, and thus, it is considered a reliable technique.

TCP uses multiple flow and error control mechanisms at transport level. In flow control, a set of segments called the window is transmitted all at once, where each segment is equipped with a sequence number. If the receiver acknowledges the highest segment, it is an indication that all the previous segments 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 [54].

In 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 a SYN bit set to 1 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 receives 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 information: the SYN bit, which is set to 1; the acknowledgment field of the TCP segment header, which is set to $ACK = x + 1$; and the server's 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 are completed, the client and server hosts can exchange data [14]. Figure 1 shows the process of three-way handshaking in TCP.

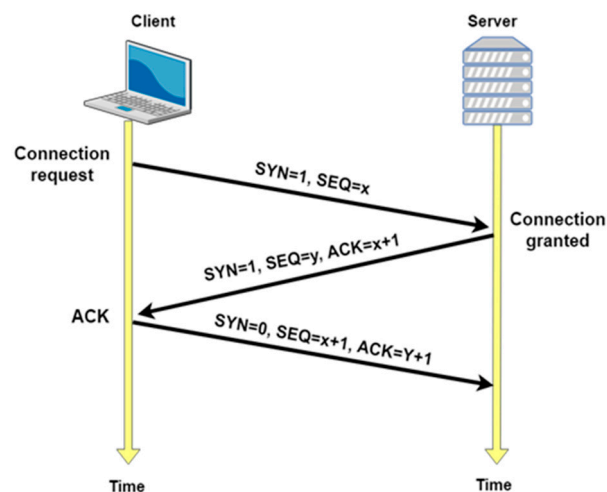


Figure 1. Three-way handshake in TCP: segment exchange.

3.1.2. UDP

UDP is considered a connectionless and unreliable protocol, as it does not require a formal handshake to initiate the data flow. Although UDP provides basic data multiplexing, it is without any reliable delivery. Unlike TCP, UDP does not establish a connection between sender and receiver. Therefore, there is no need for SYN and ACK flags to get the data flowing [55], and there is no guarantee that all 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 [56]. While these were some of the negative aspects of UDP, it is widely used in various types of applications where some data loss is accepted, such as multimedia applications [6].

UDP provides multiplexing at a low level without reliable delivery, flow, and congestion control at any network node identified by an IP address. Additionally, UDP provides checksums for data integrity and transmission errors checking. UDP also provides the port numbers used to address different functions at the source and destination of the datagram.

3.1.3. QUIC

Another protocol using unreliable techniques is the QUIC. Figure 2 illustrates the difference in the TCP and QUIC mechanisms.

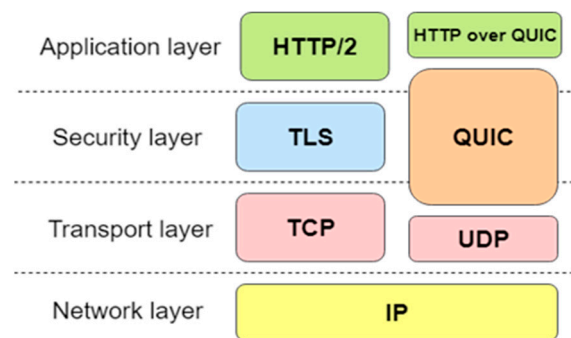


Figure 2. Traditional Quick and TCP stack.

QUIC is designed to facilitate a 0-RTT setup of connections using Diffie–Hellman (DH) exchange. The client initially sends an inchoate message to the server to obtain 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 the complete message containing its own public DH value. If the client aims for a 0-RTT, then it must not wait for the server 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 [27]. Upon a successful handshake and exchange of the ephemeral 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 config, it replies with the standard rejection message, and the handshake process can start again.

Additionally, QUIC multiplexes many QUIC streams within the single UDP connection where many responses or requests can be handled at the same UDP socket [25].

3.1.4. RDMA

Using RDMA, machines are allowed to write/read data to/from known memory regions of other network machines without involving their CPUs. 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 only supported by the switch fabric InfiniBand network, which has been expensive with no Ethernet compatibility [37]. However, many recent RDMA advances, such as RDMA over Converged Ethernet (RoCE) [37,57], are able to work over the Ethernet with data center bridging [58,59] with reasonable prices. The main difference between classic Ethernet networks and RDMA-based ones is the ability of RDMA NICs to bypass the kernel in all communications and perform hardware-based retransmissions of lost packets [38]. Based on this, the expected latency in RDMA networks is around 1 μ s, where it is around 10 μ s in classic Ethernet networks. Therefore, RDMA is widely used in datacenters, especially since the hardware has become much cheaper [34]. Figure 3 presents the QUIC handshake mechanism.

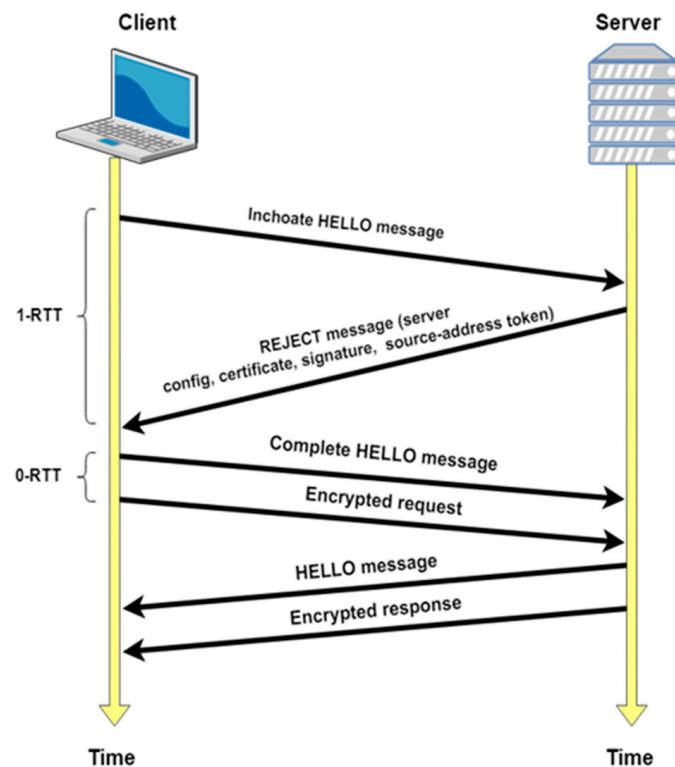


Figure 3. Initial handshaking in QUIC.

The availability of fast networks deploying RDMA technology has increased dramatically due to a noticeable drop in their prices. In fact, their prices are now comparable to the prices of the Ethernet [38]. Additionally, the abundance of cheap and large-sized dynamic random access memories (DRAMs) has played a key role in spreading RDMA technology [60]. DRAMs can be used as storage sometimes, such as open-source (BSD licensed), in-memory data structure store, used as a database, cache and message broker such as in Redis [61], or only caches such as Memcached [62]. Internally, they use many different types of data structures to provide the required memory access in the fastest way possible.

RDMA uses both transport types: connection-oriented and connectionless. Connection-oriented transport methods require a pre-defined connection between the two communicating NICs. Such connections might be considered reliable or unreliable based on packet reception acknowledgments. Unreliable connections generate no acknowledgments, which results in less traffic compared to the reliable connections. Additionally, RDMA transport through the connectionless method is implemented through the unreliable datagram, which can perform better only in one-to-many topologies [38].

In either connection method, RDMA uses a mix of transfer operations (verbs) including various types of send operations. Send operations are used for 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 invalidate. The send with Solicited event is slightly different, where the remote recipient will generate a specific event when it receives the message. A simple combination is allowed, where a message can be invalidated and can carry an event to the remote recipient with the operation Send with Solicited Event and Invalidate. To transfer data to a remote buffer, the write operation named RDMA Write is used. Similarly, to read from a remote buffer, the operation RDMA Read can be used. The final operation named Terminate is used to inform the remote recipient about an error that occurred on the sender's side using a terminate message [63]. Figure 4 shows a sample RDMA communication, as presented in [38].

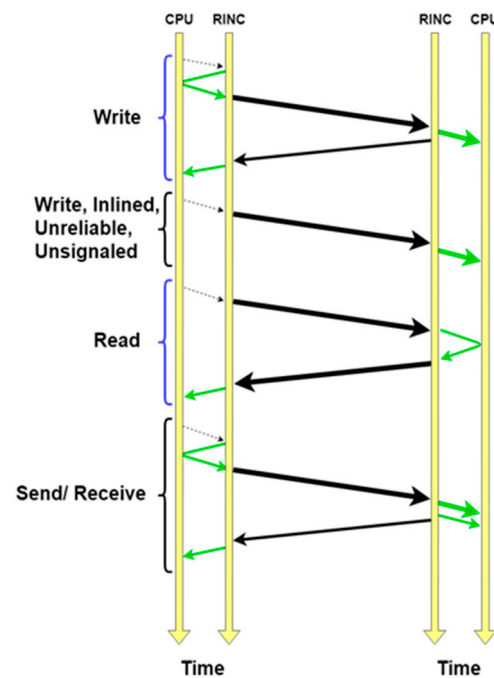


Figure 4. Steps involved in posting verbs, as presented in [38]. The dotted arrows are programmed input output (PIO) operations. The solid, straight, green arrows are DMA operations: the thin ones are for writing the completion events. The thick solid, straight, black arrows are RDMA data packets, and the thin ones are ACKs.

3.1.5. DPDK

In DPDK, a set of libraries and drivers form a lower layer with the necessary functionalities to process packets in high speed in data plane applications on Intel’s architectures. DPDK runs on 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. Figure 5 presents the DPDK communication mechanism.

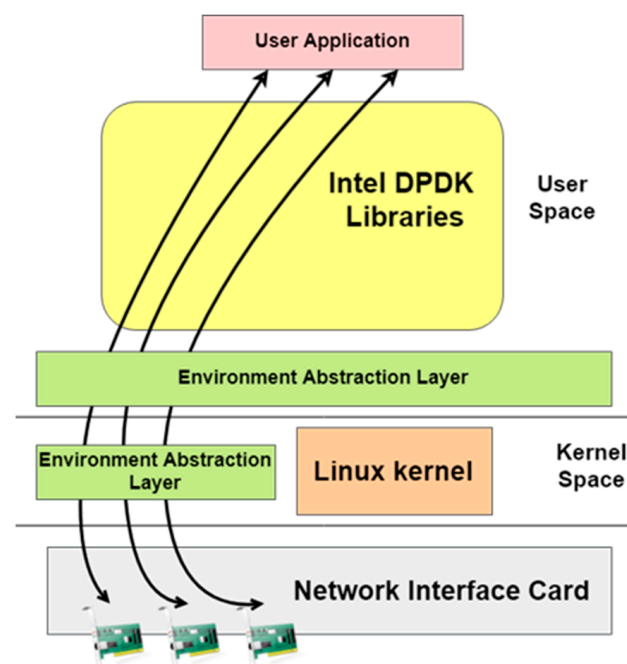


Figure 5. DPDK Communication mechanism [45].

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 [51]. 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 functionalities to handle the firewalls or protocols such as TCP or UDP that are fully supported by the Linux network stack. It is expected that programmers use available libraries to build the required functionalities.

In order to make packet processing faster, DPDK uses a set of libraries that are optimized for high performance. These libraries are executed in user space and perform all basic 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. Put simply, the libraries manage the memory, queues, and buffers as presented in the following sections.

Queue Management

The provided library manages any type of queues by providing a ring-like structure of a fixed size with doubly linked list implementation following the FIFO principle with enqueue and dequeue standard functionalities [13]. Although the lockless implementation ensures a faster writing process [64], the size of the structure is fixed and not resizable during the runtime.

Memory Management

The mapping to the physical memory is handled by the EAL [12], where every object and structure is granted a certain portion of the physical memory. To achieve maximum performance, a 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 [45].

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 afterwards. The size of the buffers is normally small, but in case larger packets are transferred, multiple buffers are chained.

DPDK uses a set of libraries to manage and allocate available buffers, which correspondingly reduces the time operating system spends in allocating and deallocating buffers. DPDK pre-allocates fixed size buffers.

The DPDK has many other side libraries performing other tasks such as the longest prefix matching (LPM) library, which helps implement a table for an algorithm to forward packets based on the IPv4 address [65]. Additionally, DPDK has also a hash library that is used for fast lookups of a specific unique key in a large set of entries [12].

3.2. Latency and Congestion Control

Latencies and losses frequently occur because of network congestions, and thus packet drops are induced. Normally, congestion control uses the so-called "Congestion Window" in order to determine the number of bytes a network can handle. However, the size of the congestion window is not standardized [66].

The TCP protocol standards remain unaware of the unexpected effects happening at network resources, including the appearance of congestion collapse [67]. In a more detailed description, TCP requires 1 RTT for the handshake and 1 or 2 RTTs in case of encrypted communication, while other protocols such as QUIC would need at most 1 RTT for the first communication and 0 RTT in case a communication occurred previously [25]. The reason behind such a 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. By eliminating kernel/user context switches, it provides low latency not exceeding 3 μ s and zero CPU overhead [34]. Additionally, RDMA provides many congestion control mechanisms, as presented in [68]. DPDK had a different way of dealing with latency, where developers using DPDK in applications proved that it could reduce the latency by up to two times compared to other available technologies [69]. Congestion is also well-managed in DPDK, using a set of supported algorithms including: tail drop, head drop, and weighted random early detection (WRED) [70]. Additionally, tasks can be conducted in stages, as cores can exchange messages [45]. On the other hand, UDP users know that it implements no congestion control [25].

3.3. Head of Line Blocking

It is a common problem in computer networks where a line of packets is held up by the first packet for various reasons. Head of line (HOL) blocking can either happen between input and output ports of routers, due to out-of-order packet delivery, or due to stream multiplexing. Since HOL blocking at routers depends on the routers' design and not on the protocol used, in this section, we focus on HOLB in the remaining scenarios.

TCP transport is prone to HOL blocking [71] due to the possible out-of-order packet delivery [25], resulting from the multiplexing of transferred data in the single byte stream of TCP's abstraction. 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 re-transmitted and received. However, in UDP, if one packet is lost, it does not stop the data from being delivered, as UDP does not enforce the in-order delivery of packets. Similarly, QUIC is designed to eliminate head-of-Line (HOL) blocking [72]. In RDMA early stage, HOL did not receive much attention, as it was considered rare to occur. However, HOL is well-handled by modern RDMA frameworks such as DCQCN [73] by introducing an end-to-end congestion control scheme. HOL might occur in DPDK too if some cores are not able to process packets fast enough, resulting in the underutilization of other cores and the drop of some packets [74].

3.4. 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 side and the receiving window at the receiver side.

The efficiency is the most important factor to tune the parameters in TCP. However, it is observed that default parameters of TCP are designed to sacrifice throughput in exchange for an efficient sharing of bandwidth on congested networks [75]. Regarding the buffer size, various techniques are used to determine the optimal size to achieve the best performance [76]. It is reported that, to achieve the best throughput in TCP, the buffer size should be dynamically adjusted to the connection and server characteristics through dynamic right-sizing [77] and buffer size allocation [78]. On the other hand, the UDP measured throughput is better than TCP under the same conditions, since UDP does not require acknowledgements, but the downside is that small fragments can be lost [75].

In modern data paths such as QUIC, an advertised connection flow limit is specified by 15 MB [27], 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% [27]. The highest throughput is achieved using RDMA, where offloading the networking stack and eliminating kernel/user context switches resulted in low latency and high throughput. In datacenters, servers achieve the highest throughput using RDMA [38].

In DPDK early stages, throughput was not on the top of the pyramid, as the bottleneck was a CPU with a limited number of cycles per second [45]. However, as the ability to

improve the performance has been recently made available to researchers, the throughput could be increased by up to 13 times [79].

3.5. Middleboxes

Middleboxes are network appliances that are capable of manipulating, filtering, inspecting, or transforming traffic, which might affect the packet forwarding. The way to deal with middleboxes differentiates old and new data paths. The TCP and UDP are long-working protocols that have existed even before most of the currently available middleboxes. Therefore, middleboxes do not affect standard TCP or UDP traffics. The TCP, for example, remains a trusted backup for modern data paths so that if their transmission is blocked through a middlebox, the transmission is switched to TCP to ensure successful transmission [27]. QUIC packets were all blocked by most middleboxes in the early stages. Later, a huge range of middleboxes was identified along with many vendors, and presently, QUIC traffic is treated reasonably in most middleboxes [27]. In RDMA, middleboxes are overlooked, as datacenters normally do not have middleboxes [80]. DPDK went further and created special versions that are applicable for middleboxes such as OpenFunction [53].

3.6. Loss Recovery and Retransmission Ambiguity

In TCP, the “retransmission ambiguity” problem was raised, as the receiver could not figure whether the received acknowledgement 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 [27]. Loss recovery is well-handled in QUIC, where the TCP “retransmission ambiguity” problem [81,82] is fully addressed and avoided. QUIC avoided this problem by assigning a unique packet number to every packet, including those carrying retransmitted data. Therefore, no further mechanisms are needed to distinguish a retransmission from an original transmission, thus avoiding TCP’s retransmission ambiguity problem.

Other protocols such as RDMA use a 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 [38]. Dropout rate was extensively studied in DPDK using many techniques in terms of packet size, hash tables, and packet buffering memory (membool). It was found that small packet sizes, complex hash tables, and extremely small or extremely large membools are associated with more dropped packets [83]. To the best of the authors’ knowledge, DPDK have no mechanism for loss recovery. On the other hand, UDP employs no loss recovery techniques.

3.7. Developer Productivity

Data paths provide the developers with multiple development or testing tools to safely use the data path. For example, in serverless communications, the Lambda model [84] can be used by developers to check responses to events such as 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 that results from browsing. Similarly, Silverlight and Flash player are excellent developer tools, but it is noticed that TCP is preferred over UDP therein [85]. However, in QUIC, testing and analyzing is provided through the developer tools of the Chrome browser, which helps measure the time elapsed since requesting a web page until the page is fully loaded. Nonetheless, RDMA and DPDK are considered more flexible where recent RDMA frameworks are customizable and developers can assign algorithms and even design the storage structure as in Derecho [86]. Similarly, the DPDK frameworks allow the developers to write their own network protocol stacks and implement a set of high-performance datagrams forwarding routines in the user space [8].

3.8. 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 [87]. The same applies to ports, where the defined port for communication is allocated for the whole conversation. Similarly, QUIC implementation was initially written with a focus on rapid feature development and ease of debugging, not CPU efficiency. Even after applying several new mechanisms to reduce the CPU cost, QUIC remained costlier than TCP [27]. On the other hand, 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 [57]. 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 a higher throughput, the goal must be to reduce the per-packet CPU cycles [45].

3.9. Target Applications

The TCP protocol fits perfectly in applications where reliable transmission is required such as emails and web browsing. However, UDP targets applications where few packet losses are allowed, such as VoIP and streaming. Streaming can also be significantly improved by incorporating the QUIC data path. The best data path for data centers would be the RDMA, where memory operations can be handled without involving the operating systems. Finally, DPDK is widely used in packet networking applications and persistently in Intel processors.

4. Simulation Findings

In this section, we present some of the simulation findings when chosen data paths were applied in environments of modern networking demands such as the long-term evolution (LTE) architectures. Simulations were preferred when traditional data paths were compared against modern data paths.

QUIC implementation for example was utilized for the network simulator ns-3 in to check the QUIC performance in LTE networks in comparison to TCP application in the same environment. The results of the simulations showed that QUIC performed better than TCP in terms of throughput and minimized latency in moderate or good transmission conditions, while they have comparable performance in poor transmission conditions [88].

Similarly, DPDK-improved middleboxes such as UDPDK showed ultra-low-latency reduction in end-to-end latency in experimental results on an LTE testbed simulating a 5G network. Such findings suggest that DPDK-improved middleboxes should have a major role in future 5G scenarios [89].

On the other hand, RDMA have been rarely used in LTE-related simulations, as it requires special hardware. However, recent emulators such as urdma might help overcome this problem in the near future [90].

To test data paths further, we performed several elementary simulations. To accomplish this part, we used iperf [91], Google Chrome developer tools, Google Chrome export tool, the net logger, and the Google Chrome Stats for Nerds. All simulations were completed using a machine equipped with an Intel 2.20 GHz CORE i7 processor and 16 GB of RAM. The results show that the number of lost packets in UDP communications widely exceeded those in TCP communications. However, using Chrome with QUIC enabled presented an impressive decrease in the number of lost packets in video streaming, although it uses a UDP substrate. This analysis was achieved by logging the Chrome usage and then streaming the videos for 3 min multiple times under the same conditions. Later, the log files were analyzed using the net-logger tool to test initiated QUIC sessions and the number of lost packets. Similarly, it was presented recently that enabling QUIC noticeably

improved the page load time [92]. Additionally, it is noticed that QUIC did not interfere in FTP file transfers that took place within Google Chrome.

5. Conclusions

In this work, we presented a thorough comparison of a set of traditional and modern data paths. We found that 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 they will probably persistently play a key role in future data transmissions. This work showed that it is useful to direct the protocol usage to some specific areas where they fit the best. RDMA fits in the communication of data centers, QUIC would help developers in multiple disciplines due to the available routines connecting with the application layer, and DPDK would work perfectly in appliances with Intel architecture.

Author Contributions: Planning, A.B., L.H. and A.G.; Analysis, A.B., L.H. and A.G.; Writing—original draft preparation, A.B., L.H. and A.G.; writing—review and editing, A.B., L.H. and A.G.; visualization, A.B., L.H. and A.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

List of Abbreviations

Transmission Control Protocol	TCP
User Datagram Protocol	UDP
network functions	NFs
Quality of Service	QoS
Deep packet inspection	DPI
Network Function Virtualization	NFV
Quick UDP Internet Connections	QUIC Protocol
Remote Dynamic Memory Access	RDMA
Data plane Development Kit	DPDK
Network Interface Card	NIC
Round Trip Time	RTT
Internet Protocol	IP
World Wide Web	WWW
The Retransmission timeout	RTO
IP Multimedia System	IMS
Real-time Transport Protocol	RTP
Internet Engineering Task Force	IETF
Network Address Translators	NATs
Domain Name System	DNS
Multipath QUIC	MPQUIC
Fast Remote Memory	FaRM
Environment Abstraction Layer	EAL
Non-Uniform Memory Access	NUMA
Advertised Window	AWND
Congestion Window	CWND
Diffie-Hellman	DH
Converged Ethernet	RoCE
Dynamic Random Access Memories	DRAMs
Longest Prefix Matching	LPM
Weighted Random Early Detection	WRED
Head of Line	HOL
Remote Procedure Calls	RPC

References

1. Khan, I.U.; Hassan, M.A. Transport Layer Protocols And Services. *Int. J. Res. Comput. Commun. Technol.* **2016**, *5*, 2320–25156.
2. Rahmani, M.; Pettiti, A.; Biersack, E.; Steinbach, E.; Hillebrand, J. A comparative study of network transport protocols for in-vehicle media streaming. In Proceedings of the IEEE International Conference on Multimedia and Expo, San Diego, CA, USA, 23–27 July 2008; pp. 441–444.
3. Andrew, T.; Wetherall, D. *Computer Networks*, 4th ed.; Prentice Hall: Hoboken, NJ, USA, 2003.
4. Saima, Z.; Faisal, B. A survey of transport layer protocols for wireless sensor networks. *Int. J. Comput. Appl.* **2011**, *33*, 45–50.
5. William, S. *Business Data Communications*; Prentice Hall PTR: Hoboken, NJ, USA, 1990.
6. Awasthi, P.; Kosta, A. Comparative Study and Simulation of TCP and UDP Traffic over Hybrid Network with Mobile IP. *Int. J. Comput. Appl.* **2013**, *83*, 9–13. [\[CrossRef\]](#)
7. Bhargavi, G. Experimental Based Performance Testing of Different TCP Protocol Variants in comparison of RCP+ over Hybrid Network Scenario. *Int. J. Innov. Adv. Comput. Sci. IJIACS ISSN* **2014**, *3*, 2347–8616.
8. Li, Z. HPSRouter: A high performance software router based on DPDK. In Proceedings of the 2018 20th International Conference, Chuncheon-si, Korea, 11–14 February 2018.
9. Luigi, R. Netmap: A novel framework for fast packet I/O. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USA, 8–10 August 2012; pp. 101–112.
10. Wu, W.; Crawford, M.; Bowden, M. The performance analysis of Linux networking—packet receiving. *Comput. Commun.* **2007**, *30*, 1044–1057. [\[CrossRef\]](#)
11. Bolla, R.; Bruschi, R. Linux software router: Data plane optimization and performance evaluation. *J. Netw.* **2007**, *2*, 6–17. [\[CrossRef\]](#)
12. Intel DPDK, Data Plane Development Kit Project. Intel. 2014. Available online: <http://www.dpdk.org> (accessed on 1 July 2022).
13. Intel DPDK, Programmers Guide. 2014. Available online: https://doc.dpdk.org/guides/prog_guide/ (accessed on 1 July 2022).
14. Kurose, J.; Ross, K. *Computer Networking: A Top Down Approach*; Addison Wesley: Boston, MA, USA, 2013.
15. Moura, G.; Heidemann, J.; Hardaker, W.; Bulten, J.; Ceron, J.; Hesselman, C. Old but Gold: Prospecting TCP to Engineer DNS Anycast (Extended). 2020. Available online: https://www.sidnlabs.nl/downloads/5OtgdybY9LK0ELbuegno38/38748ecbcabcb32e1fbc885dee3938cf/Old_but_Gold_Prospecting_TCP_to_Engineer_DNS_Ancast.pdf (accessed on 1 July 2022).
16. Paxson, V.; Allman, M.; Chu, J.; Sargent, M. Computing TCP's Retransmission Timer. 2011. Available online: <https://www.rfc-editor.org/rfc/rfc6298> (accessed on 1 July 2022).
17. Lorincz, J.; Klarin, Z.; Ožegović, J. A Comprehensive Overview of TCP Congestion Control in 5G Networks: Research Challenges and Future Perspectives. *Sensors* **2021**, *21*, 4510. [\[CrossRef\]](#) [\[PubMed\]](#)
18. Addagatla, S.; Goddanakoppalu, B.; Kumar, K.; Oyj, N. System and Method of Network Congestion Control by UDP Source Throttling. U.S. Patent 10/758,854, 21 July 2005.
19. Wirges, J.; Dettmar, U. Performance of TCP and UDP over Narrowband Internet of Things (NB-IoT). In Proceedings of the 2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS), Bali, Indonesia, 5–7 November 2019.
20. Chai, L.; Reine, R. Performance of UDP-Lite for IoT network. In Proceedings of the IOP Conference Series: Materials Science and Engineering, Sarawak, Malaysia, 26–28 November 2019; IOP Publishing: Bristol, UK, 2019; Volume 495.
21. Eggert, L.; Fairhurst, G.; Shepherd, G. UDP Usage Guidelines. 2017. Available online: <https://tools.ietf.org/html/rfc8085> (accessed on 1 July 2022).
22. Alós, A.; Morán, F.; Carballeira, P.; Berjón, D.; García, N. Congestion control for cloud gaming over udp based on round-trip video latency. *IEEE Access* **2019**, *7*, 78882–78897. [\[CrossRef\]](#)
23. Gu, Y.; Grossman, R.L. UDT: UDP-based data transfer for high-speed wide area networks. *Comput. Netw.* **2007**, *51*, 1777–1799. [\[CrossRef\]](#)
24. QUIC Working Group. IETF. Available online: <https://quicwg.org/> (accessed on 4 May 2019).
25. Carlucci, G.; Cicco, L.D.; Mascolo, S. HTTP over UDP: An Experimental Investigation of QUIC. In Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, 13–17 April 2015.
26. Viernickel, T.; Frommgen, A.; Rizk, A.; Koldehofe, B.; Steinmetz, R. Multipath quic: A deployable multipath transport protocol. In Proceedings of the IEEE International Conference on Communications, Kansas City, MO, USA, 20–24 May 2018; pp. 1–7.
27. Langley, A.; Riddoch, A.; Wilk, A.; Vicente, A.; Vicente, A.; Krasic, C.; Zhang, D.; Yang, F.; Kouranov, F.; Swett, I.; et al. The QUIC transport protocol: Design and Internet-scale deployment. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, Los Angeles, CA, USA, 21–25 August 2017.
28. QUIC, a Multiplexed Stream Transport over UDP. Available online: <https://www.chromium.org/quic> (accessed on 4 May 2019).
29. Sandvine. Global Internet Phenomena Report—Latin America and North America. 2016. Available online: https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/Internet%20Phenomena/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf (accessed on 1 July 2022).
30. Huang, S.; Cuadrado, F.; Uhlig, S. Middleboxes in the Internet: A HTTP perspective. In Proceedings of the 2017 Network Traffic Measurement and Analysis Conference (TMA), Dublin, Ireland, 21–23 June 2017.
31. Roy, F.; Reschke, J. RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. 2014. Available online: <https://datatracker.ietf.org/doc/rfc7230/> (accessed on 1 July 2022).

32. Mike, B.; Peon, R.; Thomson, M. RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2). 2015. Available online: <https://www.rfc-editor.org/rfc/rfc7540> (accessed on 1 July 2022).
33. Clark, D.; Tennenhouse, D.L. Architectural considerations for a new generation of protocols. *ACM SIGCOMM Comput. Commun. Rev.* **1990**, *20*, 200–208. [CrossRef]
34. Christopher, M.; Geng, Y.; Li, J. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In Proceedings of the 2013 USENIX Annual Technical Conference, San Jose, CA, USA, 26–28 June 2013.
35. Berk, A.; Xu, Y.; Frachtenberg, E.; Jiang, S.; Paleczny, M. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review*; Association for Computing Machinery: New York, NY, USA, 2012; pp. 53–64.
36. Kalia, A.; Mellon, C.; Kaminsky, D.G.A.M. FaST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) OSD*; USENIX Association: Savannah, GA, USA, 2016; pp. 185–201.
37. Aleksandar, D.; Narayanan, D.; Castro, M.; Hodson, O. FaRM: Fast remote memory. In *{USENIX} Symposium on Networked Systems Design and Implementation (NSDI 14)*; USENIX Association: Seattle, WA, USA, 2014; pp. 401–414.
38. Anuj, K.; Kaminsky, M.; Andersen, D. Using RDMA efficiently for key-value services. In Proceedings of the ACM SIGCOMM Computer Communication Review, Chicago, IL, USA, 17–22 August 2014; pp. 295–306.
39. Gunnar, G.E.; Eimot, M.; Reinemo, S.-A.; Skeie, T.; Lysne, O.; Huse, L.P.; Shainer, G. First experiences with congestion control in InfiniBand hardware. In Proceedings of the 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), Atlanta, GA, USA, 19–23 April 2010.
40. Jaichen, X.; Chaudhry, M.U.; Vamanan, B.; Vijaykumar, T.N.; Thottethodi, M. Fast Congestion Control in RDMA-based Datacenter Networks. In Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos, Budapest, Hungary, 20–25 August 2018; pp. 24–26.
41. Mohammad, A.; Greenberg, A.; Maltz, D.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S.; Sridharan, M. Data center tcp (dctcp). In Proceedings of the ACM SIGCOMM Computer Communication Review, New York, NY, USA, 5 September 2011.
42. Charles, L. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.* **1985**, *C34*, 892–901.
43. Kulkarni, H.; Agrawal, S.; Pore, R.; Andhale, P.; Patil, N. A survey on TCP/IP API stacks based on DPDK. *Int. J. Adv. Res. Innov. Ideas Educ.* **2017**, *3*, 1205–1208.
44. Mihai, D.; Egi, N.; Argyraki, K.; Chun, B.-G.; Fall, K.; Iannaccone, G.; Knies, A.; Manesh, M.; Ratnasamy, S. RouteBricks: Exploiting Parallelism to Scale Software Routers. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky, MT, USA, 11–14 October 2009; pp. 15–28.
45. Dominik, S. A Look at Intel’s Dataplane Development Kit. In Proceedings of the Seminars FI/IITM SS, Network Architectures and Services, Munich, Germany, 30 April–3 August 2014. [CrossRef]
46. Sangjin, H.; Jang, K.; Park, K.; Moon, S. PacketShader: A GPU-Accelerated Software Router. *ACM SIGCOMM Comput. Commun. Rev.* **2011**, *40*, 195–206.
47. Francesco, L.D.F. High speed network traffic analysis with commodity multi-core systems. In Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, Melbourne, Australia, 1–30 November 2010; pp. 218–224.
48. Nicola, B.; Pietro, A.D.; Giordano, S.; Procissi, G. On multi-gigabit packet capturing with multi-core commodity hardware. In *International Conference on Passive and Active Network Measurement*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 64–73.
49. Miguel, R.; Goutelle, M.; Kelly, T.; Hughes-Jones, R.; Martin-Flatin, J.-P.; Li, Y.-T. A map of the Networking Code in Linux Kernel 2.4. 20. *Technical Report DataTAG*. Available online: <http://files.securitydate.it/misc/draft12.pdf> (accessed on 31 March 2004).
50. Intel DPDK, Packet Processing on Intel Architecture, Presentation Slides. 2012. Available online: <https://www.dpdk.org/wp-content/uploads/sites/35/2018/03/Updated-India-DPDK-Summit-2018-MJay-NIC-Perfo.pptx> (accessed on 4 May 2015).
51. Intel DPDK, Getting Started Guide. 2014. Available online: https://doc.dpdk.org/guides-16.04/linux_gsg/index.html (accessed on 4 May 2019).
52. Intel Open Source Organization, Intel Open Source Technology Center: Packet Processing. 2014. Available online: <https://01.org/packet-processing> (accessed on 1 July 2022).
53. Chen, T.; Liu, A.X.; Munir, A.; Yang, J.; Zhao, Y. OpenFunction: Data Plane Abstraction for Software-Defined Middleboxes. *arXiv* **2016**. [CrossRef]
54. Paul, R. Selective-TCP for Wired/Wireless Networks. 2006. Available online: https://www.sfu.ca/~ljlja/cnl/pdf/rajashree_thesis.pdf (accessed on 1 July 2022).
55. Mohammad, R.; Norwawi, N.; Ghazali, O.; Faaeq, M. Detection algorithm for internet worms scanning that used user datagram protocol. *Int. J. Inf. Comput. Secur.* **2019**, *11*, 17–32.
56. Bonczkowski, J.L.; Hansen, N.A.; Hart, S.R.; Ancelot, P. 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 US-9916225-B1, 12 March 2018.
57. Alexander, S.; Zahavi, E.; Dahley, O.; Barnea, A.; Damsker, R.; Yekelis, G.; Zus, M.; Kuta, E.; Baram, D. Roce rocks without pfc: Detailed evaluation. In Proceedings of the Workshop on Kernel-Bypass Networks, Los Angeles, CA, USA, 20–25 August 2017.
58. IEEE, 802.1Qau–Congestion Notification. 2010. Available online: <https://www.ietf.org/proceedings/67/slides/tsvarea-2.pdf> (accessed on 1 July 2022).

59. IEEE, 802.1Qbb—Priority-Based Flow Control. 2011. Available online: <https://1.ieee802.org/dcb/802-1qbb/> (accessed on 1 July 2022).
60. Aleksandar, D.N.M.C.D. RDMA Reads: To Use or Not to Use? *IEEE Data Eng. Bull.* **2017**, *40*, 3–14.
61. Redis: An Advanced Key-Value Store. Available online: <http://redis.io> (accessed on 1 July 2022).
62. Memcached: A Distributed Memory Object Caching System. Available online: <https://memcached.org/> (accessed on 1 July 2022).
63. Renato, R.; Metzle, B.; Culley, P.; Hilland, J.; Garcia, D. A Remote Direct Memory Access Protocol Specification. 2007. Available online: <https://tools.ietf.org/html/rfc5040> (accessed on 1 July 2022).
64. Lwn.Net. A Lockless Ring-Buffer. 2014. Available online: <https://lwn.net/Articles/340400/> (accessed on 1 July 2022).
65. Christian, B. *Understanding Linux Network Internals*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2006.
66. Badach, A.; Hoffmann, E. *Technik der IP-Netze -TCP/IP incl. IPv6*, Hanser. 2007. Available online: <https://www.hanser-elibrary.com/doi/book/10.3139/9783446410893> (accessed on 1 July 2022).
67. Alexander, A.; Tilley, N.; Reiher, P.; Kleinrock, L. Host-to-host congestion control for TCP. *IEEE Commun. Surv. Tutor.* **2010**, *12*, 304–342.
68. Lopes, K.; Augusto, D.; Monteiro, S.; Florissi, D. Traffic Management in Isochronets Networks. In *Managing QoS in Multimedia Networks and Services*; Springer: Boston, MA, USA, 2000; pp. 131–146.
69. Chen-Nien, M.; Huang, M.-H.; Padhy, S.; Wang, S.-T.; Chung, W.-C.; Chung, Y.-C.; Hsu, C.-H. Minimizing latency of real-time container cloud for software radio access networks. In Proceedings of the 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), Vancouver, BC, Canada, 30 November–3 December 2015.
70. Organization DPDK, Documentation. 2019. Available online: <https://doc.dpdk.org/guides/index.html> (accessed on 9 April 2019).
71. Michael, S.; Kiesel, S. Head-of-line Blocking in TCP and SCTP: Analysis and Measurements. In Proceedings of the IEEE GLOBECOM Technical Conference, San Francisco, CA, USA, 27 November–1 December 2006.
72. Hamilton, R.; Iyengar, J.; Swett, I.; Wilk, A. QUIC: A Udp-Based Secure and Reliable Transport for HTTP/2. 2016. Available online: <https://datatracker.ietf.org/doc/html/draft-tsvwg-quic-protocol-02> (accessed on 1 July 2022).
73. Zhu, Y.; Eran, H.; Firestone, D.; Guo, C.; Lipshteyn, M.; Liron, Y.; Padhye, J.; Raindel, S.; Yahia, M.H.; Zhang, M. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 523–536. [CrossRef]
74. Radisys. Radisys Leads with DPDK Performance-Benchmark Study. 2014. Available online: <https://www.radisys.com/benchmark-study-radisys-leads-dpdk-performance-feature> (accessed on 5 April 2019).
75. Giannoulis, S.; Antonopoulos, C.P.; Topalis, E.; Athanasopoulos, A.; Prayati, A.; Koubias, S.A. TCP vs. UDP Performance Evaluation for CBR Traffic on Wireless Multihop Networks. *Simulation* **2009**, *14*, 43.
76. Brian, T. TCP Tuning Guide for Distributed Applications on Wide Area Networks. *Usenix SAGE Login* **2001**, *26*, 33–39.
77. Eric, W.; Feng, W.-c. Dynamic right-sizing: A simulation study. In Proceedings of the Tenth International Conference on Computer Communications and Networks, Scottsdale, AZ, USA, 15–17 October 2001; pp. 152–158.
78. Amit, C.; Cohen, R. A dynamic approach for efficient TCP buffer allocation. *IEEE Trans. Comput.* **2002**, *51*, 303–312.
79. Lockwood, J.W. Scalable Key/Value Search in Datacenters. In Proceedings of the 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, Vancouver, BC, Canada, 2–6 May 2007.
80. Gandhi, R. Improving Cloud Middlebox Infrastructure for Online Services. Ph.D. Thesis, Purdue University, West Lafayette, IN, USA, 2016.
81. Phil, K.; Partridge, C. Improving round-trip time estimates in reliable transport protocols. *ACM SIGCOMM Comput. Commun. Rev. USA* **1987**, *17*, 2–7.
82. Lixia, Z. Why TCP timers don't work well. *ACM SIGCOMM Comput. Commun. Rev. USA* **1986**, *16*, 397–405.
83. Wu, X.; Li, P.; Ran, Y.; Luo, Y. Network Measurement for 100Gbps Links Using Multicore Processors. In Proceedings of the 3rd Innovating the Network for Data-Intensive Science (INDIS2016), Salt Lake City, UT, USA, 13–16 November 2016.
84. Hendrickson, S.; Sturdevant, S.; Harter, T. Serverless Computation with Openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*; USENIX Association: Denver, CO, USA, 2016; p. 80.
85. Carol, D.; Johnston, A.; Singh, K.; Sinnreich, H.; Wimmreuter, W. SIP APIs for voice and video communications on the web. In Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications, Chicago, IL, USA, 1–2 August 2011.
86. Ken, B.; Behrens, J.; Jha, S.; Milano, M.; Tremel, E.; Renesse, R.V. Groups, Subgroups and Auto-Sharding in Derecho: A Customizable RDMA Framework for Highly Available Cloud Services. 2016. Available online: http://www.cs.cornell.edu/projects/Quicksilver/public_pdfs/Derecho-API-v9.5.pdf (accessed on 1 July 2022).
87. Shinae, W.; Park, K. Scalable TCP Session Monitoring with Symmetric Receive-Side Scaling. 2012. Available online: <https://www.ndsl.kaist.edu/~{kyoungsoo}/papers/TR-symRSS.pdf> (accessed on 1 July 2022).
88. Kyratzis, A.; Cottis, P. QUIC vs. TCP: A Performance Evaluation over LTE with NS-3. *Commun. Netw.* **2022**, *4*, 12–22. [CrossRef]
89. Lai, L.; Ara, G.; Cucinotta, T.; Kondepudi, K.; Valcarenghi, L. Ultra-Low Latency NFV Services Using DPDK. In Proceedings of the 2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Virtual Conference, 9–11 November 2021.
90. Pm-04-Dublin-Urdma-Presentation. Available online: <https://www.dpdk.org/wp-content/uploads/sites/35/2018/10/pm-04-dublin-urdma-presentation.pdf> (accessed on 1 July 2022).

-
91. Dugan, J.; Estabrook, J.; Ferbuson, J.; Gallatin, A.; Gates, M.; Gibbs, K.; Hemminger, S.; Jones, N.; Qin, F.; Renker, G.; et al. iPerf: TCP/UDP Bandwidth Measurement Tool. Available online: <https://iperf.fr/v> (accessed on 4 May 2019).
 92. Kakhki, A.M. Measuring QUIC vs. TCP on Mobile and Desktop. 2018. Available online: <https://blog.apnic.net/2018/01/29/measuring-quic-vs-tcp-mobile-desktop/> (accessed on 5 May 2019).