IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 20, 2018, accepted November 16, 2018,
date of publication February 7, 2019, date of current version February 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2886707

Keyed User Datagram Protocol: Concepts and Operation of an Almost Reliable Connectionless Transport Protocol

NUNO M. GARCIA[1,2], FÁBIO GIL[1,3], BÁRBARA MATOS[1,4], COULIBALY YAHAYA[5], (Member, IEEE), NUNO POMBO[1,2], (Member, IEEE), AND ROSSITZA IVANOVA GOLEVA[6]

[1]Departamento de Informática, Faculdade de Engenharia, Universidade da Beira Interior, 6200-001 Covilhã, Portugal
[2]Instituto de Telecomunicações, 6200-001 Covilhã, Portugal
[3]ECATI, Universidade Lusófona de Humanidades e Tecnologias, 1749-024 Lisbon, Portugal
[4]Department of Artificial Intelligence and Systems Engineering, Faculty of Computer Science and Information Technology, Riga Technical University, LV-1658 Rīga, Latvia
[5]Université de Ségou, Ségou, Mali
[6]Department of Informatics, New Bulgarian University, 1618 Sofia, Bulgaria

Corresponding author: Nuno M. Garcia (ngarcia@di.ubi.pt)

This work was supported by Grant FCT UID/EEA/50008/2013.

ABSTRACT Departing from the well-known problem of the excessive overhead and latency of connection oriented protocols, this paper describes a new almost reliable connectionless protocol that uses user datagram protocol (UDP) segment format and is UDP compatible. The problem is presented and described, the motivation, the possible areas of interest and the concept and base operation modes for the protocol named keyed UDP are presented (here called KUDP). Also, discussed are some of the possible manners in which the KUDP can be used, addressing potential problems related with current networking technologies. As UDP is a connectionless protocol, and KUDP allows for some degree of detection of loss and re-ordering of segments received out-of-sequence, we also present a proposal for a stream reconstruction algorithm. This paper ends by mentioning some of the research issues that still need to be addressed.

INDEX TERMS User datagram protocol, almost reliable protocol, data transmission statistics in UDP, algorithms for data traffic.

I. INTRODUCTION

When a user sits at a computer and, *e.g.*, using a popular communication software, starts chatting with another user who sits at another computer in another continent, computer networks provide the technologic platform that allow the information coded in that data to be conveyed from one point of the globe to the other [1], independently of the type of the machine that is at each end of the communication. The characters typed by a user at the keyboard are formatted and inserted successively into different envelopes until the final envelope is sent to the other user using the copper (or wireless) and optical fibre cables that connect all the computers that are hooked to the Internet. If all goes well, at the other end, the envelopes are successively opened and the characters the first user typed are delivered to the chat window.

At some point inside the first user's computer, one of these envelopes will be formatted according to the Transmission Control Protocol [2], making this an envelope a TCP segment, or, following the Open Systems Interconnection (OSI) notation, an OSI Layer 4 [2] Protocol Data Unit (PDU). Although there are several protocols that can be used at the transport layer / layer 4 level, the most common are the TCP and the UDP, this latter standing for User Datagram Protocol [2].

The major difference between the way TCP and UDP work, stands on the warranties that each protocol gives to the user or application, regarding the effective and ordered delivery of the content from one end to the other of the communication network. While TCP guarantees that all segments will be delivered in the order they were created, and if not, the source machine will be notified, UDP sends all segments using a best-effort approach, *i.e.*, if the network is overloaded and the packets containing the UDP segments are dropped at an overloaded or malfunctioning router, neither the transmitting nor the receiving machines will be aware of that. In this case, the user who was trying to engage in a live video conversation with another user who is at another point of the Internet (*e.g.*), will perceive the bad quality of the call as consequence of the faulty/incomplete data transmission, this scenario being classified as faulty data transmission that results in poor quality of experience.

Because TCP includes mechanisms [2] that allow for the sequencing and acknowledgment of sent and received packets, individually or in dynamic or static transmission windows, to simplify a more complex set of mechanisms that several flavors of TCP use (and that are not the subject of this paper), it is only natural to conclude that a communication using TCP has a greater overhead than a similar UDP communication would have [2]. On the one hand, this issue is never a real problem, because when a user wants to transmit a file from one machine to another, or chat with another user, or download a web page, all the packets have to be sent and received at the correct order or reordered at the destination machine, because if one single packet is lost or two packets are received in switched order, that will render the file unusable and maybe unreadable, or possibly cause the conversation on the chat to end up being interpreted by the person on the other side in a different manner that it was intended to be. Yet on the other hand, if one is doing a video conference, one can afford to lose or reverse the order of arrival of a couple of packets, and most of the times this is irrelevant to the user's perceived quality of the experience of the transmission.

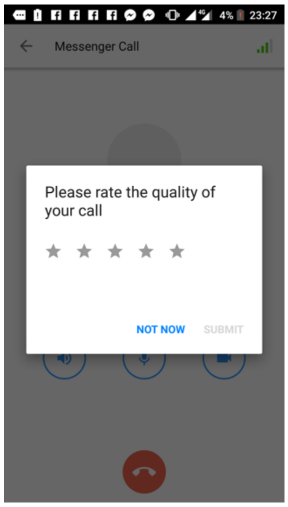
But for scenarios where the received data must mimic in perfection the transmitted data, the mechanism that TCP uses to start a ''conversation'' between two machines may be just too heavy for networks that have scarce resources. Or, if a network has a low bandwidth, the TCP protocol will adapt by creating more frequent acknowledgment packets, resulting from smaller transmission windows [2], increasing the overhead of the communication. This is also why at Layer 4, we can consider that a communication of data occurs in a Stream for TCP transmissions, but for UDP, data communication occurs in a segment*per* segment basis, as there is no manner to establish an order relation among the individual segments that constitute the initial set of packets that were formed as a response to a data communication request.

Such is the case of sensor networks, typically consisting in networks of disperse devices with power, network and transmission constrained capabilities, called nodes, who communicate with *e.g.* a central node in order to provide readings of the sensors in these peripheral devices to the data storage and processing host machine. For example, if a sensor must periodically send small files or pieces of data to a central machine, maybe missing one file over a period of several readings is not critical to the purpose of the system. Yet, it would be useful to know if a received file is usable or not, not being usable meaning that maybe the packets were received in a different order they were sent, or maybe a packet is missing from the series of packets that formed the initial file transmission set.

There are other examples that may benefit from a fast transmission of packets between two machines, including chat applications, as long as the receiving application is able to inform the user that some text is missing or that a garbled message has been received.

For example, with the increase in number and in size of sensor networks, but also the increase of the need to transmit multimedia contents, it is expectable to see an increase in these two traffic profiles, one consisting of many short streams of very small packets, and others consisting of many long streams of larger packets (yet packets will still be smaller than 1500 bytes, a limit that is imposed by payload size from the underlying Ethernet frame [3]).

Or, in another scenario, it would be extremely useful if popular video call applications (*e.g.* Skype, Whatsapp or Facebook) could automatically assess or infer the quality of experience of any given call. For example, after closing a Facebook Messenger video call, the application shows a feedback dialog window where the user is expected to rate how well the communication was experienced, and what the user felt went wrong (please see Figure 1 and Figure 2).

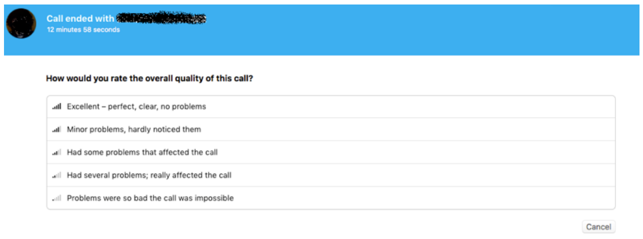


Please rate the quality of your call

★ ★ ★ ★ ★

NOT NOW SUBMIT

FIGURE 1. **Feedback request dialog box shown after a Facebook messenger video call.**



Call ended with [redacted]
12 minutes 58 seconds

How would you rate the overall quality of this call?

- Excellent – perfect, clear, no problems
- Minor problems, hardly noticed them
- Had some problems that affected the call
- Had several problems; really affected the call
- Problems were so bad the call was impossible

Cancel

FIGURE 2. **Feedback request dialog box shown after a Skype video call.**

For all of the above examples, but also to other examples where some type of data imputation [4] could be done to *e.g.* replace the missing data in lost packets, or when the platform needs to know the success metrics of the transmission, it would be desirable that one could use an almost reliable zero overhead type of transport protocol.

The purpose of this paper is to propose a new communication protocol based on standard UDP, that configures an almost reliable, zero overhead mean to transmit data between two machines by allowing some degree of connection-oriented features in a connectionless transmission, foreseeing its applicability in sensor networks and in real time video and audio applications.

Yet, as the goal of this paper is also to also invite researchers to further extend this concept, this paper does not present comparative results for this new protocol, being this a subject of future, hopefully, cooperative work. This new protocol is named Keyed User Datagram Protocol (KUDP; we choose to read it as ''Keyed UDP'').

The remainder of this paper is organized as follows: this paragraph concludes Section 1 where an introduction to the topic was presented; Section 2 presents a brief history and state of the art for TCP and UDP data transmissions; Section 3 presents with detail the new KUDP and discusses the operating manners in which the KUDP may be used, and Section 4 presents a proposal the KUDP reconstruction algorithm and some results. Section 5 concludes the paper with the appropriate remarks. As this paper is focused on presenting and discussing the new KUDP, the extended simulations that will confirm or infer its merits will be addressed in future work, being also the intention of this paper to put KUDP forward to open discussion by the research community, with the goal of attracting momentum for the creation of a Request For Comments (RFC).

II. TCP AND UDP TRANSMISSIONS: A QUICK REVIEW

This section briefly addresses the TCP and UDP protocols, its operation, and also previous attempts made to solve the problem of lowering the TCP communication overhead while still keeping full control over the transmission outcome. As these protocols are well-established from a long time, this section will not discuss these with detail but will instead introduce them briefly and point the interested reader to further reading, as much of the significant research occur for specific types of networks such as e.g. mobile networks or data centers [5].

The Transmission Control Protocol (TCP) [6] is the golden standard for reliable communications between different devices. TCP insures reliability and integrity of data transmissions because of the mechanisms it implements, namely, the Three-Way Handshake, that is responsible to establishing connections of TCP, the Additive Increment Multiplicative Decrement (AIMD) and others. Problems with TCP have been discussed in extensive literature, and the interested reader can refer to [1], and [7]–[12].

TCP exchanges control packets between the two communicating machines to acknowledge and control the flow of a window of packets, and resizes this windows in response to overload in communication channels. It also uses a time-out mechanism to detect packet loss [13].

Being the first example of a Connection-Oriented protocol because it simulates a connection between the sender and the receiver, and as a result of the way TCP is designed, TCP has become one of the most prevalent transport protocols in networks [2], despite the number of messages it needs to transmit.

Opposed to TCP, the connectionless User Datagram Protocol (UDP) [2] does not guarantees the reliability and integrity of data transmission. UDP has been mostly used in for real-time applications that demand low latency and can afford some packet loss and out-of-sequence receptions such as video and voice over IP (VoIP) [14].

Recognizing the high latency of TCP and the unreliability of UDP, researchers have suggested a significant number of protocols that try to reach a compromise between the overhead in communications generated by the need of control and the willingness to, in some scenarios, afford to lose some data.

The Stream Control Transmission Protocol (STCP) [15] aims to be a general purpose transport protocol similar to TCP but able to take advantage of the features of modern IP networks, such as multihoming. For STCP multihoming means that it will try to use the addresses that are associated with different network interface cards present in one single host to improve the data transmission efficiency. Yet, STCP adopts congestion control and set up and tear down procedures that are similar to those of TCP and therefore it does not diminish the overhead relative to TCP, on the contrary, as its control procedures are more complex. Chellaprabha *et al.* [12] tested the performance of TCP, UDP and STCP for wireless sensor networks and concluded that given the negligible packet loss for these types of networks, UDP performs best in terms of average throughput of data for the studied scenarios.

Acknowledging the slow start and the unfairness of shorter Round-Trip Time (RTT) transmissions relative to longer RTT in TCP, Gu *et al.* [16] and Gu and Grossman [17] proposed the UDP with congestion control and acknowledgement mechanisms, also called UDT. The data and the control messages exchanged use two pairs of ports at the sender and receiver machines, and the protocol considers timers, acknowledgments and congestion control mechanisms among other techniques. As it adds control messaging to UDP it configures a higher overhead protocol than UDP.

Considering the specific needs of real-time control systems, the poor latency of TCP and the unreliability of UDP, CRETP (for Conditional Retransmission Enabled Transport Protocol) [18] proposes the use of UDP accompanied by retransmission and acknowledgment features, by using timers that detect, for a given real-time data transmission, if a given packet must be retransmitted or not.

Malhotra *et al.* [19] developed an UDP based chat application that exchanges acknowledgments after each message. Here the focus was not a real-time application but rather to limit the overhead induced by TCP when the data window is composed of a single small data packet.

Aiming to reach a compromise between the unreliable UDP and the high latency TCP, but being aware that video transmission does not have to be 100% perfect, Porter and Peng [20] suggest a hybrid protocol, where TCP is used to transmit the most important packets of data, and UDP is used to the other packets, reporting an improvement in the performance of the transmission even at moderate loss rates.

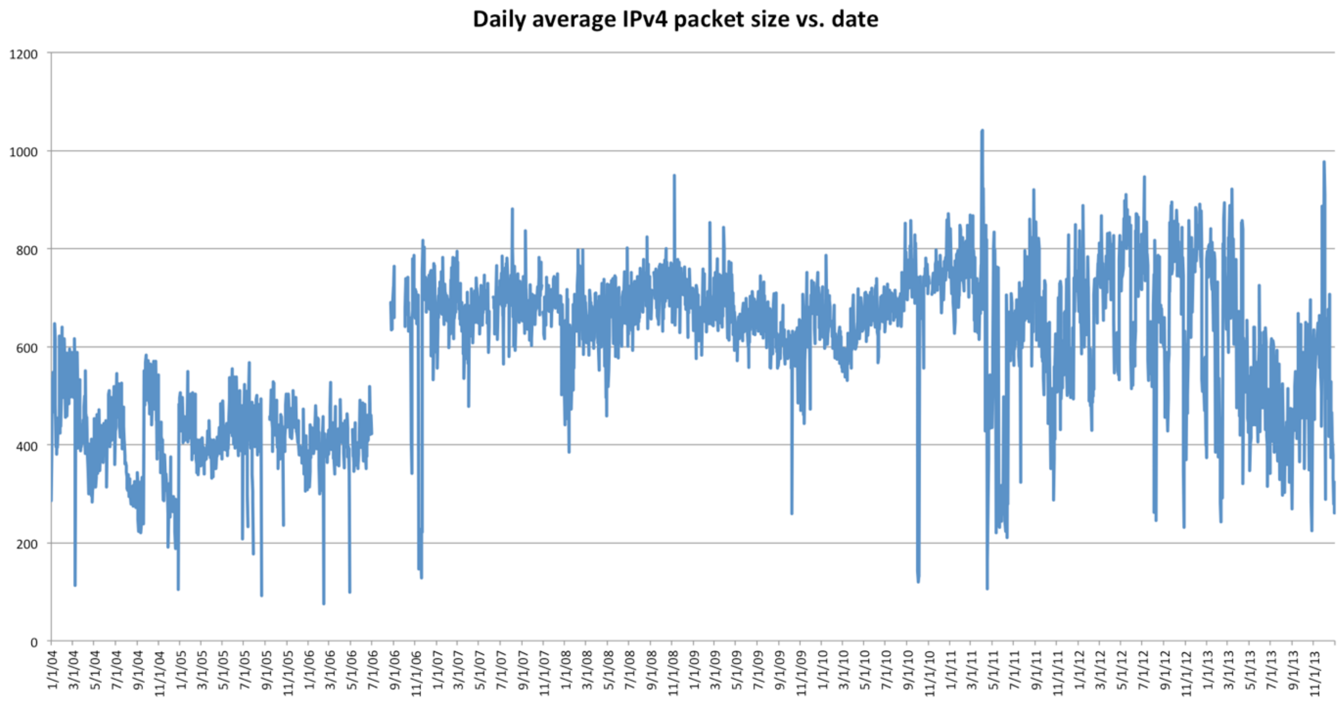


FIGURE 3. Average IPv4 daily packet sizes, between 1/1/2004 and 31/12/2013 (in Bytes).

A class of reliable UDP-based adaptive transport protocols [21] (RUNAT) was proposed. With the goal of achieving a high throughput at the application level, the protocol uses a rate control scheme founded on the stochastic approximation method. RUNAT relies on messages fed back from the receiver to the sender to adapt its transmission throughput as to generate flows that, according to the authors, are friendly to TCP traffic. By using feedback messages to notify the sender, RUNAT increases the overhead of transmission when compared to pure UDP and is unable to resolve out-of-sequence arrivals.

Comparing to RUNAT, older protocol SABUL [22] uses UDP to transmit data and TCP to convey control messages and uses special message formats to achieve high data throughput in grid applications.

Assessing and comparing the performance of four types of UDP protocols, the Yue *et al.* of [23] show that RUBDP, Tsunami, UDT, and PA-UDP conclude that PA-UDP performs the best. The authors also briefly describe these protocols: RUBDP uses TCP to acknowledge the reception of all packets in a stream; Tsunami relies on negative acknowledgments sent by the client as to packets that did not arrive; and PA-UDP implements a three-way-handshake to start its data transmission (UDT was discussed previously in this section). From this analysis stems that all these protocols need to use control messages to implement its specific features.

III. KEYED USER DATAGRAM PROTOCOL

This section describes the operation of the KUDP. It starts by identifying the problem, and proceeds with the description of the manner KUDP will be used. This section also describes how the port keys can be defined and how to address the problems that KUDP may present. As to simplify the language, it will be assumed that KUDP is already implemented, and therefore the verbs in the sentences will be used in its present form.

A. THE PROBLEM

Figure 3 shows the daily average packet size for IPv4 packets, along a 10-year time frame, between 2004 and 2014. This data was collected from a Japan-USA link, by the MAWI project [24], and the chart shows three distinctive areas (or maybe five), of which the first two are relevant for the purpose of this paper.

In order to better understand this figure, a brief explanation of the collection process is in order. The collection refers to a daily 15 minutes recording of Internet traffic passing in a link connecting Japan and the USA, between 14:00 and 14:15 hours Japanese standard time. This period of time often contains between 300k and 500k unique IP addresses. It is therefore assumed that the traffic is ergodic, also because it contains communications between academic institutions and the rest of the Internet. At sample point B, congestions were often observed, as this was an 18Mb/s link, with a 100Mb/s committed access rate. MAWI reports that in July 2006, the overloaded link B was replaced by the overprovisioned link F, which started as a full 100Mb/s link and was upgraded to a bandwidth of 150Mb/s on the 1st of June 2007. For more information on the naming of the collection points, please see [25].

Observing Figure 3, it can be seen that between 2004 and July 2006, the average packet size is around 400 bytes, and after the link change, shown as a disruption in plotted data, the average packet size jumped to over 600 bytes. In fact, these are the two main areas of Figure 3 that interest this research, being the others, eventually, the following: the third one, between the November 2011 and January 2013, depicting a higher variability on the average packet size, the fourth one, between January 2013 and August 2013, showing again another decrease in the average packet size, and possibly, the fifth and final one, between August and December 2013, showing an increase in the average packet size. Yet, these last observations are not to be considered as they are just visual observations and also because these are not relevant for the purpose of this paper.

Taking **Figure 3** into consideration, it has to be asked how is it possible that there is a 66% increase in the average packet size, from 404 bytes to 670 bytes, regarding the two previous and the two consecutive years of changing the link, particularly because the change in the average packet size can be observed immediately before and after the upgrade of the link.

Also, as the generation of traffic is done by applications at the host computers, there is no ground to assume that all or a large majority of the users have changed their traffic generating applications in the short amount of time needed to change from an overloaded link to an overprovisioned link. On the contrary, it can only be assumed that the traffic generation applications in the tributary computers did not changed significantly over that period of time, let alone changed drastically in the course of a few months in 2006.

As a base research hypothesis, one must assume that over this period of time, the application layers at all the tributary host computers have continued to generate payloads that have resulted in IP packets whose size is only determined by the nature of the tributary applications themselves and limited by the payload size of the underlying Data Link layer, usually the 1500 Bytes MTU imposed by the 30 plus year old Ethernet. And as we have no data that can support or refuse this base hypothesis, one has to rely on the knowledge of the software ecosystem of the time to conclude that the in fact there was no massive change in the nature and type of the users' applications that could have produced an 66% increase in the average packet size over a short period of time.

If layer 7 did not change its nature in the tributary computers, the reason for the change in the average packet size needs to be researched at the lower layers. Yet, it is not expected nor plausible that the Physical, nor the Data Link, nor the Internet layers suffered any change because of increase of capacity of the link; at the host level, the Physical layer concerns solely the transmission of bits over the medium and is completely oblivious about link underloads or overloads; the Data Link layer typically imposes the Internet Maximum Transmission Unit (MTU) as 1500 bytes by means of the Ethernet standard frame maximum payload size, and is also oblivious for network load conditions; finally, the Internet layer handles IP addressing, and has no mechanism to detect, prevent or circumvent link overload.

After ruling out all the possible sources for this behavior, and also because there were no other changes of this magnitude observed in this period [26], and taking in consideration that this metric is an average packet size, *i.e.*, is the average of the size of all packets captured in this time-frame, it has to be considered that the packets that were transmitted by the tributary machines are not only the ones generated by the applications at layer 7, but also the packets that are generated at these same machines by the protocols as *e.g.* TCP control messages.

This is to say that the increase in the average packet size was very likely caused because fewer very small packets were transmitted, *i.e.*, there was less intervention of the transmission mechanisms for the TCP protocol, that detect and prevent transmission errors by adjusting not only the maximum segment size, but more importantly, the size of the transmission windows, fitting these to the capacity of the link, to assure that all packets in a transmission were either transmitted with success or its errors dully acknowledged.

As a conclusion, it has to be assumed that in the first period of this observation, between early January 2004 until early summer 2006, it was the generation of many small packets resulting from TCP transmission control mechanisms that lowered the average packet size, and these machine/protocol generated packets were the fact that caused the observed behavior.

Moreover, as a side conclusion of this paper and following this line of reasoning, the hypothesis that overloaded links tend to show smaller packet size averages needs to be further researched. It can also be assumed that these changes in the average packet sizes will happen for both IPv4 and IPv6. Unfortunately, it's not possible to assess this assumption for IPv6 because at this time there was still not enough IPv6 captured traffic.

B. A KEYED UDP PROTOCOL

As an introductory note, let us define KUDP as an extended UDP protocol that is implemented in network protocol stack in the source and/or destination hosts at layer 4 and that KUDP data segments are standard UDP segments. Similar to UDP, KUDP is an end-to-end unicast protocol.

In the remainder of the paper, we will use of some freedom referring to packets and segments. When we say that packets are sent from port 10000 (*e.g.*) we mean that packets were sent from the IP address of the source machine and contained a segment that was generated at port 10000. We take this liberty of language because although segments correspond to ports, segments are not transmitted *per se*, but packets are.

Consider the following sample UDP transmission: at the source machine, an ordered set of ten UDP segments encapsulated in the corresponding packets is sent from *e.g.* port 10000. While the source port will be a random port that the application managing the transmission has acquired because it was available, usually, the destination port at the destination

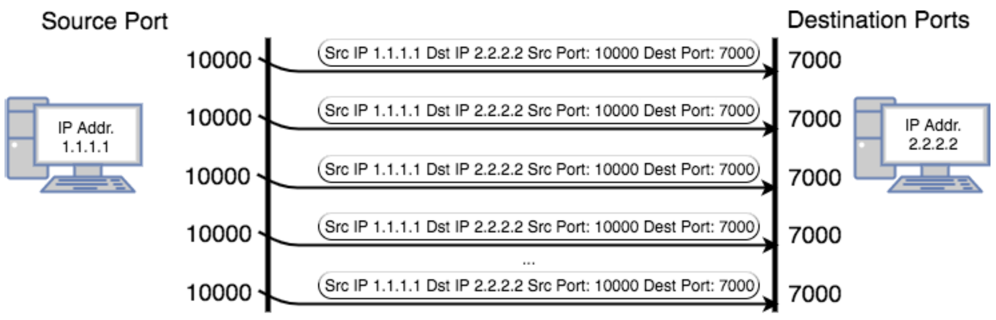


FIGURE 4. Scheme depicting the sending of an ordered set of several packets between two machines, using UDP.

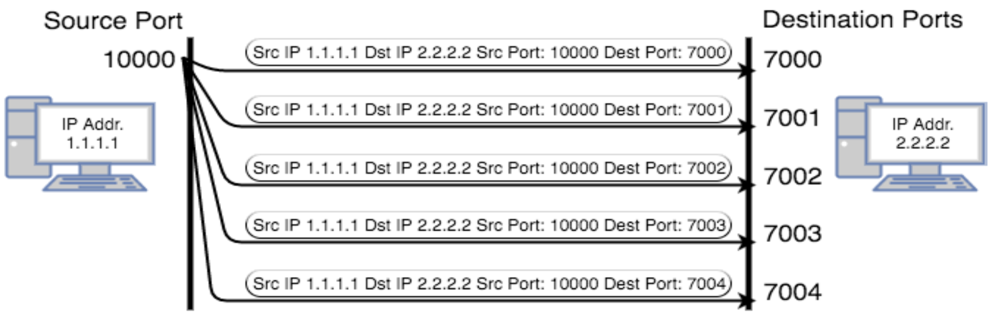

Source Port
Destination Ports
10000
IP Addr. 1.1.1.1
IP Addr. 2.2.2.2
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7000
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7001
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7002
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7003
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7004
7000
7001
7002
7003
7004

FIGURE 5. Scheme depicting the sending of an ordered set of 5 packets between two machines, using Keyed UDP.

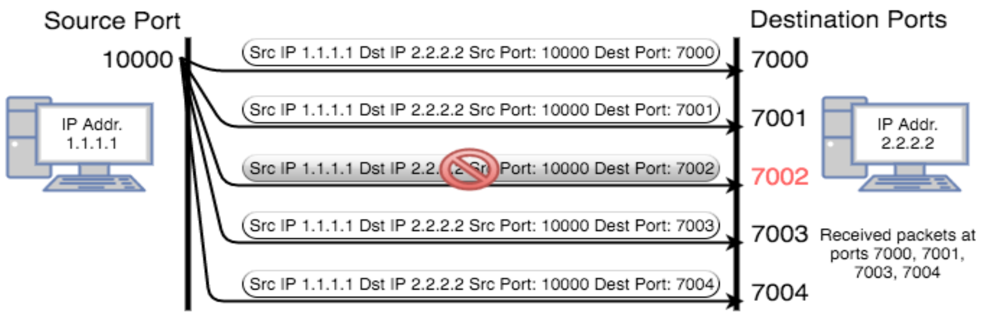

Source Port
Destination Ports
10000
IP Addr. 1.1.1.1
IP Addr. 2.2.2.2
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7000
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7001
Src IP 1.1.1.1 Dst IP 2.2. Port: 10000 Dest Port: 7002
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7003
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7004
7000
7001
7002
7003
7004
Received packets at ports 7000, 7001, 7003, 7004

FIGURE 6. Basic loss sample scheme, depicting a transmission with a lost packet.

machine will be fixed and predetermined, *e.g.*, port 7000. At the destination machine, a server software application will have previously reserved, opened and started listening to port 7000 as a way to be able to receive all the incoming packets, extract its payload and proceed to do whatever it is meant to do, *e.g.* to forward the payload to the video chat window (see Figure 4). Using UDP, the destination machine has no way to assess if all the transmitted packets were received, or if there were packets that have arrived out of sequence.

Moreover, if the received packets do not have the same order in which they were sent, the destination machine has no manner to re-establish the initial order of the received packets, because the UDP segment header has no field in which to record the sequence number in which the packets were generated at the source machine.

C. DESTINATION KEYED UDP

Now consider that the source machine sends the ten packets to the destination machine, but instead of sending all the segments to *e.g.* port 7000, sends the segments to *e.g.* ports 7000, 7001, 7002, 7003, 7004, and again 7000, 7001, 7002, 7003, and finally 7004. Let us also momentarily consider that these port numbers are somehow hardcoded or defined as parameters into the application software both at the client and at the server side. At the destination machine, the application software has reserved and opened ports 7000 up to 7004 and expects to receive packets in all these ports. Figure 5 depicts the transmission of the first five packets between these two machines, arbitrarily addressed (figure shows IPv4 for convenience of example).

As the reception machine knows the expected reception sequence, this information allows this machine to assess

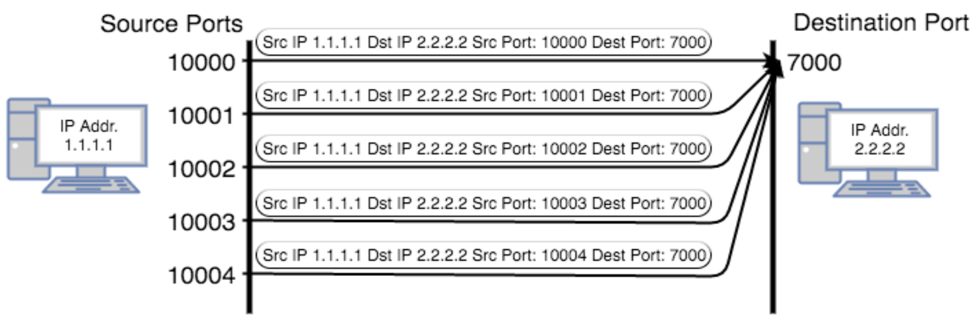

Source Ports
10000
10001
10002
10003
10004
IP Addr. 1.1.1.1
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10000 Dest Port: 7000
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10001 Dest Port: 7000
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10002 Dest Port: 7000
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10003 Dest Port: 7000
Src IP 1.1.1.1 Dst IP 2.2.2.2 Src Port: 10004 Dest Port: 7000
Destination Port
7000
IP Addr. 2.2.2.2

FIGURE 7. **Source keyed UDP sample transmission.**

two scenarios: basic loss and basic out-of-sequence. In this example, the server software is listening to a set of 5 ports, therefore the length of the key is defined as 5, *i.e*, in a single side keyed UDP transmission, the length of the key is n if $n+1$ is sum of the number of ports that will be used at the transmission by both sides. In a double KUDP transmission n is the length the key if the transmitting side is using s ports, and the receiving side is using d ports and $s + d = n$.

D. SOURCE-DESTINATION-KEYED UDP AND TCP/IP PROTOCOL STACK

It is also possible to conceive a double keyed UDP, or sdKUDP for source-destination KUDP, where the source sends packets from a range of ports to another range of ports at the destination, possibly with different port range lengths. It is conceivable that, known by both applications, a sequence of packets is sent from, *e.g.*, a set of three source ports to a set of five destination ports.

The proposal of the KUDP can correspond to the insertion of an additional sublayer at the transport level of the TCP/IP protocol stack, as shown in Figure 8. The UDP layer is replaced by the KUDP and the Stream Reconstruction Algorithm, as these two sublayers work together to implement the features of KUDP.

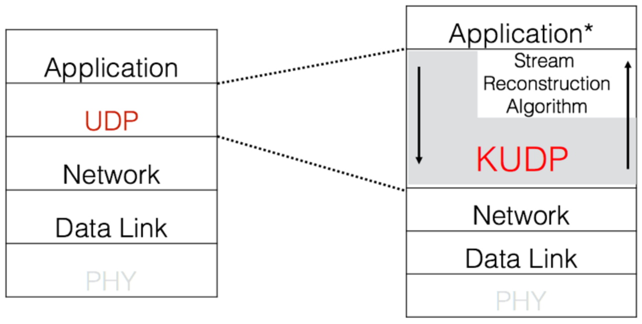


FIGURE 8. **Sub-layering the transport layer at the TCP/IP protocol stack.**

In the descending direction, the application layer communicates directly with the KUDP layer, *i.e.*, the KUDP layer receives the data from the application layer and performs the UDP tasks with the additional port keying activity. In the ascending direction, after the packets have been received from layer 3 and passed to the KUDP layer, the Stream Reconstruction Algorithm will attempt to rebuild the original data stream. The application layer is referred as "Application∗" as the applications may be KUDP aware, in which case the Stream Reconstruction Algorithm will provide statistics and control data to the application.

If this is not the case, and the transport layer is to remain unchanged, meaning that the applications must use standard UDP in all cases, then the application itself must encode all the features of the KUDP and manufacture the segments as to use standard UDP at the lower layers.

E. NON-CONSECUTIVE PORT KEYS

Finally, it is also conceivable that the packets are send from or to a list of non-consecutive ports, or, following another pre-defined key, *e.g.*, packets are sent from or to ports 10000, 10000, 10001, 10001, 10005, 10003, as if each port corresponds to a music note and there is a song to be played. This feature can prove to be more complicated to implement because of active Network Address Translation (NAT-PT) policies, but let us keep in mind that NAT-PT machines may alter the source IP address and source port, but not the destination address nor the destination port, although the destination address may be changed by some IPv4 to IPv6 translation mechanisms (or vice-versa) [27]. These and other potential problems will be subject of additional research.

For all the three scenarios, source destination keyed UDP, source keyed UDP or destination keyed UDP, it is possible to use a single port, or a range of ports, or list of ports, not necessarily consecutive. The manner in which the packet flow is addressed from and/or to these ports is referred to as the transmission port key.

As a summary, and to simplify the description of the initial concept, as long as the sending and receiving applications know in advance what is the port key *i.e.* what are the source and/or destination ports, the communication can be established with some features that allow for basic loss detection and/or out-of-sequence detection and correction.

F. MUTUAL DEFINITION, AGREEMENT OR DISCOVERY OF THE TRANSMISSION PORT KEY

First of all, we have to consider the different manners in which the source and destination applications know from and/or to which ports they should transmit to and/or receive data packets.

As with every other application that needs to transmit data over the Internet or over a computer network in general, the client application has the information on which port the server application is listening, *i.e.*, the destination port is often hard coded into the client and server applications. Such is the case of standard HTTP communications [2] and many others that use the Well-Known port list [2].

For KUDP we can have two basic scenarios. The first one consists in a previously agreed and hardcoded definition of the port key into the client and server applications. In this scenario, the port key is said to have been determined. A determined port key has different implications for the sKUDP, dKUDP and SDKUDP. The second scenario consists in using a communication agreement protocol that allows the source and the destination applications to define and agree on the manner and on the port key for the KUDP communication. In this scenario, the port key is said to have been agreed. A third non-basic scenario considers keys that were not determined nor agreed, but rather are discovered.

Let us consider hardcoded port keys for dKUDP. In this case, the source application will know beforehand to which ports it will be transmitting and the destination application will have previously reserved and opened the predefined set of ports. After receiving the packets and performed reordering, detecting faults, and eventually, doing packet data imputation, the destination application can therefore proceed to send the corresponding payloads to the upper layers.

Considering hardcoded keys for the sKUDP gives us again two possible scenarios, because the destination application may or may not know beforehand from which ports it will be receiving the data packets. Let us assume that the destination application knows beforehand that it will be receiving packets from ports $\{p_1, p_2, p_3, \ldots, p_k\}$ and let us assume that these ports are not changed by means of a Network Address Translation/Port Translation (NAT-PT) or other mechanisms that the packets may encounter while traveling in the network. In this case, the key is still said to have been determined and the destination application will receive packets whose segments are identified as having been transmitted from the $\{p_1, p_2, p_3, \ldots, p_k\}$ ports and proceed to interpret these accordingly, implementing the loss and out-of-sequence detection algorithms.

It is also possible that the destination application does not know beforehand from which ports it will receive the UDP packets. Again, let us consider that the source application decides to send segments from the ports $\{p_1, p_2, p_3, \ldots, p_k\}$. The destination machine will then be able to infer what is the source key once it has received the first two ports that are repeated in the sequence, *i.e.*, once it has received packets from at least ports $\{p_1, p_2, p_3, \ldots, p_{k,}p_1\}$. This is considering that there were no errors in the first transmission and that the port key is a trivial port key, or more specifically, that the first port is not repeated in the port key. For non-linear non-trivial port keys, *i.e.* for port keys that are not composed by a linear sequence of non-repeated ports, or for cases where there have been errors in the initial transmission of the segments, the port key and the port key length can still be inferred in a dynamic manner but at the cost of additional computation. In the case that the destination application does not know beforehand what is the port key, the key is said to have been discovered.

For hardcoded sdKUDP the joint application of the above-mentioned procedures is still valid.

The second scenario includes a setup where the source and destination applications need to agree on the port key before the actual data transmission begins. Although this will be subject of additional research, it can be considered that it is possible to establish one communication on a specific port, and that by means of this communication, the source and destination applications will agree on the mutual port key or keys, in the case of a sdKUDP data transmission. This will of course mean an additional overhead in the communication, but it will also bring advantages for data transmission in networks where ports are blocked or are scarce. This setup is termed Key Definition Protocol and will be subject of further research.

Table 1 summarizes the data transmission scenarios. Each line corresponds to a type of KUDP, each column corresponds to a port key definition scenario and each cell describes the means by which the port key is defined.

TABLE 1. Means of definitions for port keys in KUDP.

Key is / Protocol is	**Determined**	**Discovered**	**Agreed**
dKUDP	Hardcoded	n.a.	Key Definition Protocol
sKUDP	Hardcoded	Inference	Key Definition Protocol
sdKUDP	Hardcoded	Inference	Key Definition Protocol

G. RETRO-COMPATIBILITY WITH STANDARD UDP APPLICATIONS

It is possible that the destination application in a sKUDP data transmission can still receive data packets that were transmitted from a standard UDP source application, as the formatting of the segments still adheres to the UDP RFC [14].

If the source application is standard UDP and if sKUDP is used, the destination KUDP application will receive a set of packets that are sent from a single source port, therefore making it impossible to assess loss or out-of-sequence events. If the source application is standard UDP and if dKUDP is used, the destination KUDP application will receive a set of packets that are sent to a single destination port, also making it impossible to assess loss and out-of-sequence events.

For sKUDP, if the source KUDP application sends packets to a non-KUDP destination application, this will almost be the for a standard UDP communication, as the destination application will still receive a set of packets at a single destination port.

For dKUDP, if the source KUDP application sends packets to a non-KUDP destination application, the $n-1$ packets in each set of n packets will be lost because the source application does not know and cannot confirm that the packets sent to *e.g.* ports 10001, 10002, 10003 and 10004 will not be received because the destination application is expecting packets in just *e.g.* port 10000.

Table 2 summarizes the results of data transmission in sKUDP and dKUDP when either the source application or the destination applications use standard UDP data transmission methods.

TABLE 2. UDP to KUDP communication.

Protocol / The UDP applic. is…	**dKUDP**	**sKUDP**
Source application is standard UDP	Destination application receives all packets to a single destination port, not allowing for out-of-sequence and loss event detection.	Not applicable.
Destination application is standard UDP	Only every nth packet will be received because the source application will try to send packets to destination ports that are not being monitored by the destination application.	This is almost the case for the standard UDP scenario, so no loss nor out-of-sequence event detection is possible. If the receiving application does not check the source ports for the incoming packets, the transmission will be perceived as UDP.

H. KEYED IPv6

One of the possible adaptations of KUDP is to not consider the use of keyed sets of ports, but rather to use a range of IPv6 addresses that are configured at the appropriate interface of the sender and/or receiver machine. In fact, a hybrid model using different IPv6 and different ports can be devised. One can imagine that from an observer point of view, such a stream of packets would look like different communications between different applications in different machines, more so if the correspondent payloads were to be encrypted.

We named this the Keyed-IPv6 and all previous and following considerations still apply, adapting the concept from port to IP Address. It may happen that some operating systems may not allow the assignment of several IPv6 addresses to the same interface [28], although this is technically feasible. Keyed-IPv6 will likely overload ARP tables at the machines in its respective local networks, although this has to be further investigated.

I. ADDITIONAL CONSIDERATIONS

Additional issues deserve extended research, starting from what can be done once the destination application identifies a faulty transmission because of missing data packets. Again, there are different possible scenarios. The trivial one is to do nothing but to keep statistics on the data transmission quality, informing the upper layers and/or the monitoring applications of the found errors. A non-trivial one would be to perform data imputation. In fact, having the few previous packets and the few following packets, imputation can be performed at the application layers, or as soon as the payloads of the packets are interpreted. This can make some sense in video or voice applications, for example. Once more, this is a new field open to further research.

Finally, it is conceivable that the destination application can signal the source application that some packets in the set have not arrived. Provided that the source application has kept a copy of the initial set, or a copy of the p last packets, the retransmission can be attempted. The signaling of the missing packets message as well as missing packets re-transmission can be done using a set of different ports used for signaling, *i.e.*, a set of ports that are used as mean to convey control data between the two applications.

Time-stamping the packets at arrival can also prove helpful while identifying loss events, port key inference and other communication events, such as *e.g.* the end of data transmission.

Complexity can be added into the protocols by way of the definition of dynamic port keys instead of the static port keys addressed in this paper, *e.g.* by using a port list that changes from one moment to the other, *e.g.* following some function of time or number of packets sent. Additionally, the sending application can intentionally garble the original message by sending out-of-sequence packets that later can be reordered at the destination machine.

One possible problem may rise from the existence of Network Address Translation (NAT) machines in the middle of the communication. NAT machines often perform also Port Translation (NAT-PT) [29] in both IPv4 and IPv6 [2]. A typical scenario happens when a NAT-PT machine receives a packet with source address *e.g.*192.168.1.1 and destination address 193.165.66.1 and source and destination ports 7000 and 10000 respectively. To achieve its purpose, the NAT-PT machine will remanufacture the packet in a way that the source IP address and port address may be changed, *i.e.*, the packed that is transmitted into the network after the NAT-PT machine may have source IP address 193.165.6.2 and port address 7890 and keep the original destination IP and port addresses, while it will maintain a translation table where it records at least the information that at date-time μ the pair address-port 192.168.1.1 and 7000 was transmitted as 193.165.66.2 and 7890. As the NAT-PT machine does not alter the destination IP and port addresses, the dKUDP port key is not altered by the action of NAT-PT machines.

As long as the source and destination address spaces are translated with homomorphic functions that are persistent over the time adequate to complete a data streaming transmission, sUDP and sdUDP will still be able to work. A potential problem will be the increase on the length of the translation tables for NAT-PT machines, as one set of KUDP data transmission will increase the number of entries from one to n, where n is the length of the port key.

Multicast in KUDP can be implemented as the multicasting feature is implemented at the 3rd layer of the protocol stack. Therefore, a stream of packets coming sent to a multicast address would still be replicated at the routers, independently of the destination ports.

We will also have to address issues like security from at least two perspectives: the first one is what new vulnerabilities will this new protocol bring forward, and the second one is, the integration of encryption in KUDP. Finally, we can also use this method to measure latency and jitter in UDP; departing from the assumption that the packets at the source are transmitted at a constant rate, we can measure the delays between packets and the variability of these delays.

IV. A PROPOSAL FOR THE STREAM RECONSTRUCTION ALGORITHM FOR KUDP

This section presents a proposal for a stream reconstruction algorithm for KUDP. The algorithm will be discussed by presenting on an example of a dKUDP transmission with key length 6, considering both packet loss and packet switching. The following example consists in a transmission from one port to a sequence of six ports, numbered 1 to 6 for simplification purposes. This sequence consists in a series of 30 packets, sent to ports 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, … and so on. Also for simplification purposes, please note that the number of packets to be sent are a multiple of the size of the key.

In this example, we considered a scenario where 5 packets were lost and 5 packets arrived out-of-sequence. For clarity of this example, we numbered the packets as 1a, 2a, 3a, 4a, 5a, 6a, 1b, 2b, 3b, 4b, …, 1e, 2e, 3e, 4e, 5e and 6e, just to allow us to keep track of the behavior and results of the algorithm. In this numbering scheme, 1, 2, 3, 4, 5, 6 are the port numbers and *a, b, c, d* and *e* are the first round of packets, the second round of packets and so on. As it can be seen, the algorithm considers only the number of the port in its sorting procedures.

For this example, we have the following transmitted (*Ts*) and received sets (*Rs*):

Ts = {*1a, 2a, 3a, 4a, 5a, 6a, 1b, 2b, 3b, 4b, 5b, 6b, 1c, 2c, 3c, 4c, 5c, 6c, 1d, 2d, 3d, 4d, 5d, 6d, 1e, 2e, 3e, 4e, 5e, 6e*}

Rs = {*1a, 3a, 4a, 6a, 5a, 2b, 4b, 6b, 3b, 2c, 3c, 1c, 4c, 6c, 1d, 2d, 4d, 3d, 5d, 6d, 1e, 3e, 4e, 6e, 5e*}

As it can be seen from the above example, 30 packets were transmitted and 25 were received. Randomly market for loss are packets *2a, 1b, 5b, 5c* and *2e*, marked in red. Also randomly market for out-of-sequence reception and marked in yellow background, are packets *5a, 3b, 1c, 3d* and *5e*. The later reception of these packets is 1, 2, 2, 1 and 1 respectively, meaning that that packet *5a* was delayed and arrived after 1 packet in the sequence, packet *3b* was delayed and arrived after 2 packets in its sequence and so on.

At the receiving machine, the algorithm builds a buffer that is 5 packets long, *i.e.* it builds a $n-1$ long buffer, where n is the size of the key. The value for $n-1$ was found in an empirical manner, as a window of size n or $n+1$ do not offer additional information for the algorithm and windows of size smaller than $n-1$ do not allow the construction of a significant election set. Nevertheless, this is something that needs to be researched with additional detail, as other sizes of sorting windows and other approaches have not been attempted.

This buffer is updated each time a packet arrives, as Figure 9 shows, and for each set of received packets, a newly sorted vector of packets is produced.

Figure 9 shows the first 10 iterations for this 25 packets stream.

If the transmitting stream has k packets, then k iterations will be made. Please note that these iterations can be made as soon as the nth packet arrives, *i.e.* the algorithm can run in real time with an initial delay of n packets.

Let us analyse some of the iterations shown in Figure 9. For iteration 1, the algorithm will already have received the first $n-1$ packets. In this case, the reception machine will have received the packets {*1a, 3a, 4a, 6a, 5a*}. The algorithm will place these packets in an array, placing all the packets in order, *i.e.,* it will create the following output {*1a, f, 3a, 4a, 5a, 6a*}. The f in position 2 is placed because the algorithm recognizes that the list it received should have a packet destined to port 2. As this packet is missing, its position is occupied with a f for *failure to receive*.

This is also de case of iteration 7, among others in the above example. Iteration 7 will look to start filling its array at position 1, as $1 = 7 \bmod n$, where n is the size of the key, 6 in our example. Iteration 7 receives the packets {*4b, 6b, 3b, 2c, 3c*}, by this order. As it is expecting a packet in position 1 and there is none in the list, its output for the first position will be f. The only packet received in port 2, the packet labeled with *2c*, will occupy the second position. The third position will be occupied with the first packet received to port 3, the packet *3b*. The 4th position will be occupied by packet *4b*, the 5th position will hold f, and the 6th position will store packet *6b*. Yet, as packet *3c* still needs to be placed, the 7th and 8th positions will hold f and packet *3c* will be stored in 9th position of the sorting vector. The algorithm will fill positions with f or with packets as long as there are packets that were received and need to be sorted. Therefore, iteration 7 will output this sorted vector: {*f, 2c, 3c, 4b, f, 6b, f, f, 3c*}. Please note that although the sorting windows has always a $n-1$ size, the sorting set will vary in size, depending on the packets it needs to place in order.

The packets can be sorted and missing packets can be acknowledged as soon as the iteration moves,

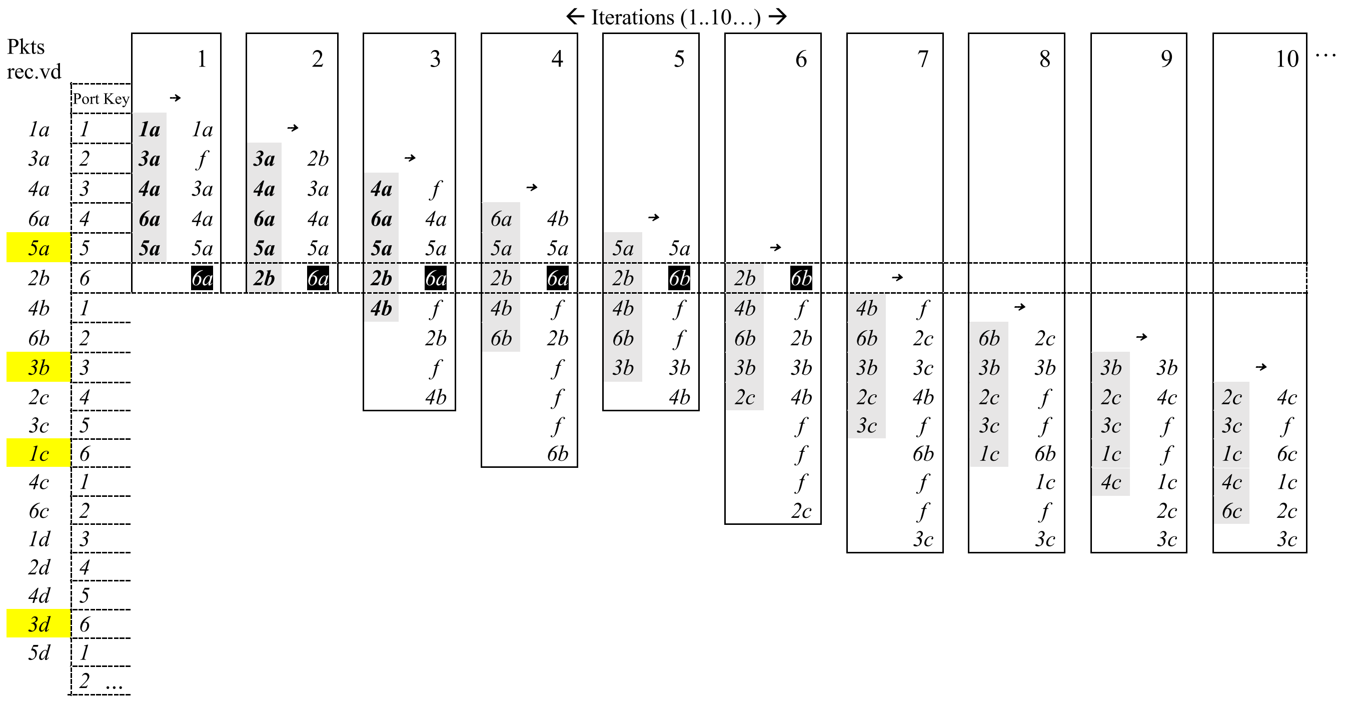


FIGURE 9. First 10 iterations and corresponding sorting sets. At the left of each iteration, the received set of packets, at the right of each iteration, the sorted set of received packets, showing eventual missing packet. In black background, the candidates for the election for port number 6.

i.e., the first received packet can be inferred after iteration 1, the second received packet can be inferred after iteration 2, and so on.

The final choice for the packet in position *i*th can be made as soon as the iteration *i*th, *i.e.*, as soon as the $i+n-1$ packet has arrived.

This choice procedure is an election made among the candidate packets in each of the previous iterations, for example, reading the sorting results line by line, one can see that the candidates for the packet in the 6th position are {*6a, 6a, 6a, 6a, 6b, 6b*} (see **Figure 9**, consider the elements in the line for 6a, shown in back background and white font color). The election procedure is detailed by example as follows:

- For the 1st packet there is only one candidate, this is packet *1a*, therefore, packet *1a* takes the first position;
- For the 3rd packet the candidates are {*3a, 3a, f*}, therefore, packet *3a* wins;
- For the 2nd position, the candidates are {*f, 2b*}. As there is a tie between *f* and *2b*, the first candidate wins, *i.e.*, the algorithm concludes that the 2nd packet never arrived;
- For the 4th position, the candidates are {*4a, 4a, 4a, 4b*}, therefore, *4a* wins.

The candidates for the remaining positions (5 to 10) are, respectively:

5th: {*5a, 5a, 5a, 5a, 5a*}, elected *5a*
6th: {*6a, 6a, 6a, 6a, 6b, 6b*}, elected *6a*
7th: {*f, f, f, f, f*}, elected *f*
8th: {*2b, 2b, f, 2b, 2c, 2c*}, elected *2b*
9th: {*f, f, 3b, 3b, 3c, 3b, 3b*}, elected *3b*
10th: {*4b, f, 4b, 4b, 4b, f, 4c, 4c*}, elected *4b.*

Please note that for any given position, the number of candidates is not deterministic, as it depends on the relative position of the sorted packets inside each iteration.

After all the iterations (not shown here), the final sorted packet set looks like this:

Fs = {*1a, f, 3a, 4a, 5a, 6a, f, 2b, 3b, 4b, f, 6b, 1c,*2c, 3c, 4c, ***5d***, 6c, *1d, 2d, 3d, 4d,* ***5d***, *6d, 1e, f, 3e, 4e, 5e, 6e*}.

In this example, the loss ratio was 5/30 (16.67%) and the out-of-sequence ratio was 5/25 (20.00%), and the tests demonstrated that it is possible to create a maximum likelihood sequence, acknowledging the packets that were effectively not received.

Given a number of candidates, the election rules are as follows:

1) the candidate with most occurrences wins;
2) in case of a tie, the first occurrence for that candidate wins;
3) in case of a hidden packet, fit the packet in the first corresponding free slot;
4) finally, once elected, a candidate cannot be eligible again, *i.e.*, no duplicates are allowed.

Hidden packets are packets that never get elected but that were received nevertheless. There are cases of hidden packets for packet loss and out-of-sequence ratios higher than 20%. In this case, the system may not recover in almost-real-time for the hidden packets because these can only be acknowledged some time before the sequence has been produced.

Also, as it can be seen in *Fs*, packet *5d*(in bold) appears as an elected candidate for both position *5c* and position *5d,* violating election rule 4. For some applications having a duplicate packet may be acceptable, so research on the impact of hidden and duplicate packets also needs to be further researched.

V. CONCLUSIONS AND NEXT STEPS

In this paper, we have presented the concept for a zero overhead almost reliable manner to transmit data packets in an UDP-like manner. The main argument for the conception of this protocol is that when networks are overloaded, the TCP control packets cause a reduction on the overall average packet size that impairs the usefulness of the scarce available bandwidth.

The definition of port key, and the manner in which the data packet transmission would occur, in a source, destination or source-destination manner was also discussed. The concepts of detection of packet loss and detection and correction of out-of-sequence reception of packets were also presented for the different protocols.

Some potential problems were discussed with particular detail the ones related to the interference that can be introduced in the communication by NAT-PT machines that are in the communication path, namely consisting of two problems: the potential increase in size of the NAT-PT translation tables, and the problems that non-homomorphic translation algorithms can cause to the correct identification of the port keys.

Also, and because the concepts in this paper are still only concepts, some lines of future research were pointed. First and foremost, the validation and simulation of the efficiency of the sKUDP, dKUDP and sdKUDP for data transmission; secondly, the variation of average packet size as a mean to assess network link overloads; thirdly, the definition of a signaling protocol that will enable the agreement of the port keys and eventually the signaling of statistics and data retransmission requests.

Resilience to non-homomorphic algorithms in NAT-PT machines is also worth of additional research, as well as the use of packet time-stamping to enhance loss and end-of-transmission events. Moreover, complexity as a security feature can be researched by means of the use of dynamic port keys.

There is work to be done regarding the definition of additional communication protocols that may bring to KUDP some of the main features of a transmission in TCP.

Also to be researched is the trade-off between the number of ports in a key and the effort posed on the Stream Reconstruction Algorithm, as increasing the number of ports allows for a greater degree of certainty on the lost and out-of-sequence packets, but it increases the effort on the reconstruction algorithm as it increases the size of the sorting sets. In the context of the definition of the control protocol, it is possible to define keys that increase or decrease dynamically in size when the quality of the communication decreases or increases, respectively.

Finally, there is also research work to be done on the redefinition / mapping of the layers at the OSI model, as to integrate the manner the KUDP extends its mechanisms to layers 4 and 7.

Recovering a paragraph from the introduction section, this paper presents the concept for a new zero overhead almost reliable protocol, also with the goal to open discussion by the research community, and of attracting the interest of researchers for the creation of a Request For Comments (RFC).

ACKNOWLEDGMENT

This article/publication is based upon work from COST Action IC1303 - AAPELE - Architectures, Algorithms and Protocols for Enhanced Living Environments and COST Action CA16226 - SHELD-ON - Indoor living space improvement: Smart Habitat for the Elderly, supported by COST (European Cooperation in Science and Technology). More information in www.cost.eu.

REFERENCES

[1] V. G. Cerf and R. E. Icahn, "A protocol for packet network intercommunication," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 71–82, 2005.

[2] B. A. Forouzan, *TCP/IP Porotcol Suite*. New York, NY, USA: McGraw-Hill, 2002.

[3] N. M. Garcia, M. M. Freire, and P. P. Monteiro, "The Ethernet frame payload size and its effect on IPv4 and IPv6 traffic," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2008, pp. 1–5.

[4] Y. Wexler, E. Shechtman, and M. Irani, "Space-time completion of video," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 463–476, Mar. 2007.

[5] A. Bakre and B. R. Badrinath, "I-TCP: Indirect TCP for mobile hosts," in *Proc. 15th Int. Conf. Distrib. Comput. Syst.*, May/Jun. 1995, pp. 136–143.

[6] J. B. Postel, Ed., *Transmission Control Protocol*, document IETF RFC 793, Sep. 1981.

[7] M. Allman, V. Paxson, and W. Stevens, *TCP Congestion Control*, document IETF RFC 2581, Apr. 1999.

[8] W. Stevens, *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, document IETF RFC 2001, Jan. 1997.

[9] M. Handley, S. Floyd, J. Padhye, and J. Widmer, *TCP Friendly Rate Control (TFRC): Protocol Specification*, document IETF RFC 3448, Jan. 2003.

[10] O. Ait-Hellal and E. Altman, "Analysis of TCP vegas and TCP reno," *Telecommun. Syst.*, vol. 15, nos. 3–4, pp. 381–404, 2000.

[11] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[12] B. Chellaprabha, D. S. ChenthurPandian, and D. C. Vivekanandan, "Performance of TCP, UDP and SCTP on sensor network with different data reporting intervals," *IOSR J. Eng.*, vol. 2, no. 4, pp. 621–628, 2012.

[13] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, 1998.

[14] J. B. Postel, *User Datagram Protocol*, document IETF RFC 768, Aug. 1980.

[15] T. Dreibholz, E. P. Rathgeb, I. Rungeler, R. Seggelmann, M. Tuxen, and R. R. Stewart, "Stream control transmission protocol: Past, current, and future standardization activities," *IEEE Commun. Mag.*, vol. 49, no. 4, pp. 82–88, Apr. 2011.

[16] Y. Gu and R. Grossman. *Using UDP for Reliable Data Transfer Over High Bandwidth-Delay Product Networks*. Accessed: Nov. 22, 2018. [Online]. Available: https://www.researchgate.net/publication/248287123_Using_UDP_for_Reliable_Data_Transfer_over_High_Bandwidth-Delay_Product_Networks

[17] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, 2007.

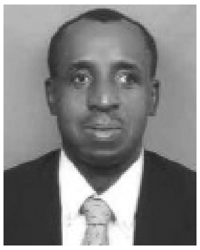
[18] L. Gui, Y.-C. Tian, and C. Fidge, "A conditional retransmission enabled transport protocol for real-time networked control systems," in *Proc. IEEE 36th Conf. Local Comput. Netw. (LCN)*, Oct. 2011, pp. 231–234.

[19] A. Malhotra, V. Sharma, P. Gandhi, and N. Purohit, ''UDP based chat application,'' in *Proc. 2nd Int. Conf. Comput. Eng. Technol. (ICCET)*, Apr. 2010, pp. V6-374–V6-377.
[20] T. Porter and X.-H. Peng, ''HYBRID TCP/UDP video transport for H.264/AVC content delivery in burst loss networks,'' in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2011, pp. 1–5.
[21] Q. Wu and N. S. V. Rao, ''A class of reliable UDP-based transport protocols based on stochastic approximation,'' in *Proc. 24th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, Mar. 2005, pp. 1013–1024.
[22] Y. Gu and R. Grossman, ''SABUL: A transport protocol for grid computing,'' *J. Grid Comput.*, vol. 1, no. 4, pp. 377–386, 2003.
[23] Z. Yue, Y. Ren, and J. Li, ''Performance evaluation of UDP-based high-speed transport protocols,'' in *Proc. IEEE 2nd Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Jul. 2011, pp. 69–73.
[24] (Dec. 1, 2012). *MAWI (Measurement and Analysis on the WIDE Internet) Working Group Traffic Archive*. [Online]. Available: http://mawi.wide.ad.jp/mawi/
[25] K. Cho, K. Mitsuya, and A. Kato. *Traffic Data Repository Maintained by the MAWI Working Group of the WIDE Project*. [Online]. Available: http://mawi/.wide.ad.jp/mawi
[26] M. Karikari, ''Analysis of Web protocols evolution on Internet traffic,'' M.S. thesis, Dept. Comput. Sci., Univ. Beira Interior, Covilhã, Portugal, 2014.
[27] P. Wu, Y. Cui, J. Wu, J. Liu, and C. Metz, ''Transition from IPv4 to IPv6: A state-of-the-art survey,'' *IEEE Commun. Surveys Tuts.*, vol. 15, no. 3, pp. 1407–1424, 3rd Quart., 2013.
[28] R. M. Hinden, ''IP next generation overview,'' *Commun. ACM*, vol. 39, no. 6, pp. 61–71, 1996.
[29] G. Tsirtsis and P. Srisuresh, *Network Address Translation-Protocol Translation (NAT-PT)*, document IETF RFC 2766, Feb. 2000.

NUNO M. GARCIA received the B.Sc. degree (Hons.) in mathematics/informatics, and the Ph.D. degree in computer science and engineering from the Universidade da Beira Interior (UBI), Covilhã, Portugal, in 2004 and 2008, respectively. He was the Founder and also the Coordinator of the Assisted Living Computing and Telecommunications Laboratory, a research group within the Instituto de Telecomunicações, UBI. He was also the Co-Founder and is currently the Chair of the Executive Council of the BSAFE LAB–Law enforcement, Justice and Public Safety Research and Technology Transfer Laboratory, a multidisciplinary research laboratory in UBI. He is the Coordinator of the Cisco Academy, UBI. His main interests include next-generation networks, algorithms for bio-signal processing, distributed and cooperative protocols, and the battle for a free and open Internet. He is a member of the ISOC, and the Non-Commercial Users Constituency, a group within GNSO in ICANN. He was the Chair of the COST Action IC1303 AAPELE–Architectures, Algorithms and Platforms for Enhanced Living Environments, Brussels, Belgium.

FÁBIO GIL received the B.Sc. degree in computer science engineering. He is currently pursuing the M.Sc. degree in computer science engineering with the Universidade da Beira Interior. He is currently a Quality Assurance and Deploy Automation Analyst with Readiness IT-Systems Integration, Fundão, Portugal. He is also a Junior Researcher with the Instituto de Telecomunicações.

BÁRBARA MATOS received the B.Sc. degree in computer science engineering. She is currently pursuing the M.Sc. degree with the Universidade da Beira Interior. She is currently a Junior Researcher with the Instituto de Telecomunicações. She is also a Junior Researcher with the Department of Artificial Intelligence and Systems Engineering, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia. She is a member of the QAD-Quality Assurance and Deployment at Readiness IT-Systems Integration, Fundão, Portugal.



COULIBALY YAHAYA received the B.E. degree (Hons.) in computer and information engineering from International Islamic University Malaysia, Malaysia, the M.Sc. degree in networks and communication engineering from Universiti Putra Malaysia, and the Ph.D. degree in computer science from Universiti Teknologi Malaysia (UTM). He was with AGETIC, a government agency in charge of public ICT development (infrastructures and applications), under the Ministry in Charge of ICT, Bamako, Mali, from 2004 to 2008. During this period, he occupied the post of the Director of Infrastructure and Development, from 2006 to 2008. In 2012, he was with UTM, as a Visiting Lecturer. Furthermore, he was a Senior Lecturer with UTM, from 2012 to 2015. He is currently the Head of the Research and Development at AGETIC, and also an Assistant Professor with the Université de Ségou, Ségou, Mali. His main research interests include optical communication and networks, wireless and mobile networks, and the Internet of Things. He has authored numerous journal and conference papers. He is a member of the IEEE Communication Society and Optical Society of America. He is also a referee for many high impact journals and conferences, including the IEEE COMMUNICATION LETTERS, the *IEEE Communication Magazine*, the *International Journal of Communications Systems* (Wiley), and *Sensors*.

NUNO POMBO is currently an Assistant Professor with the Universidade da Beira Interior (UBI), Covilhã, Portugal. He is the Coordinator of the Ambient Living Computing and Telecommunication Laboratory, UBI. His current research interests include: information systems (with special focus on clinical decision support systems), data fusion, artificial intelligence, and software. He is also a member of BSAFE Lab, the Instituto de Telecomunicações, UBI, and the COST Actions: CA15109 European cooperation for statistics of network data science.

ROSSITZA IVANOVA GOLEVA received the M.Sc. degree in computer science and the Ph.D. degree in communication networks. She has experience in teaching for more than 15 different courses, had long and short scholarships in Denmark, Greece, Portugal, The Netherlands, Israel, and Germany. She has 36 years of experience as a Professor. She is currently an Assistant Professor with the Department of Informatics, New Bulgarian University, Sofia, Bulgaria. She has authored over 100 research papers in various conferences and journals, and has edited three books. She has participated in many EU funded and regional projects, including EU ECHO Project ASpires (2017–2019) on forest fire prevention and detection, IoT-based home automation, smart environment systems, IC1303 COST Action algorithms, architectures and platforms for enhanced living environments, and IC0703 COST action traffic and monitoring analysis. She is dealing with the design and implementation of hybrid environments, quality-of-service analysis, high-performance cloud computing, and traffic engineering and simulations. She serves as a reviewer of European commission research programs, many conferences, and journals. She has been the Vice Chair of the IEEE Bulgaria Section, since 1999, where she is currently leading the Communication Chapter.