Possibilities in network transport protocols to audio stream application context

Flávio Luiz Schiavoni
Department of Mathematics and Statistics
University of São Paulo
São Paulo - Brazil
Email: fls@ime.usp.br

Abstract—Audio streams have always been seen as a kind of network application that don’t care about the packet loss or data integrity. In the same way, it always appears as a latency-sensitive application. However, different scenarios may be mapped from different uses of these resource onto a network data distribution. Although many applications are using UDP protocol, there are some possibilities of transport protocol to the this scenario. Thus, this paper will present four different transport protocol on the TCP/IP stack: TCP, UDP, STCP and DCCP. Theoretical comparison will present the protocol features and a minimal protocol transport socket implementation as a Pure Data external will be presented to send multi channel audio through network. This research stands implementation of Medusa - a distributed sound environment.

I. INTRODUCTION

With the growth in the use of computer networks and its specific use to networked audio and music, the desire of share real time audio have been demanding research and investigations from the both interested sides: musicians and network engineers. Many computer network books present the audio stream as a typical case of use of UDP [1] transport protocol, sometimes attached with RTP [2] or RTSP [3] application protocol. The use of these protocols in audio stream application may be seen on OSC [4], NetJack [5], SoundJack [6], JackTrip [7], [8], elamming [9], Otherside [10] and LDAS [11].

The network bibliography use to cite that a lost data is not a crucial impairment to audio streaming because it might result only in a small glitch in the played-out audio/video [12]. Other common point is that multimedia requires high bandwidth what is impossible to reach with a full integrity data transportation [13]. Even with all the improvements in the network technology nowadays, it is still impossible to control the balance between real time data transfer and packet loss in huge multimedia network distribution, regardless it is a common wish.

Better stream networking can be reached if the balance between latency and packet loss could tilt to one side depending on the audio application scenario. For instance, a musical practice using networks should not worry about packet loss but latency, but a musical recording audience should care about packet loss but latency [14].

This scenario approach invites us to review the way of distribute audio through computer networks claiming that other transport protocols can be used instead of only UDP [15]. TCP [16] is a transport protocol that is reliable and its uses can bring less packet loss than UDP. Other transport protocols that will be investigated is DCCP [17] [18] and SCTP [19].

Although many books suggest the RTP protocol to send multimedia streams, RTP is not a transport protocol but application protocol made as an addendum to UDP with a timestamp and a sequence number in each datagram. RTP always appears with RTCP (Real Time Control Protocol), a control protocol that is used together with RTP to provide a kind of reliable transport. Other application protocol used to stream audio media is RTSP [3] (Real Time Stream Protocol). RTSP is considered a good solution to streaming on demand application because it has implemented a protocol with commands like START, PAUSE and TEARDOWN.

The work presented in this paper is made as investigation to stand the development of Medusa [20], a distributed audio environment. The source code used in the practical part of this paper will be available in the Medusa project site 1.

The remainder of this paper is organized as follows. Section 1 presents the 4 network protocols involved in this paper, Section 2 presents the Pure Data implementation. Conclusion and future works are presented in section 3 and 4.

II. BACKGROUND

This section presents the transport protocols involved into this research.

A. TCP/IP protocol stack

The TCP/IP protocol suite is always expanding its utility and it is seen as an unfinished system [21]. It is divided into layers for division of labor and ease of new alternative layer implementation [22]. The layers of the TCP/IP protocol are presented on Fig. 1

![Figure 1. The layers of the TCP/IP protocol stack. [12]](http://sourceforge.net/projects/medusa-audionet/)

1http://sourceforge.net/projects/medusa-audionet/
Modern operating systems have a separation between users process and the operating system processes. Applications protocol are user processes while transport protocol and below layers are normally provided as part of the operating system kernel [23]. It means that any new application protocol can be created and implemented by anyone while a transport protocol should be provided by the operating system and its development and deployment will need a super user privilege.

The whole idea about TCP/IP stack is that any layer can be changed or reimplemented without breaking the data coupling. So, the application does not need to know how TCP is implemented or which device driver will be used to transport the data.

B. Internet Protocol (IP)

IP is the workhorse protocol of the TCP/IP protocol suite [24] because it is used to pack almost all users data in TCP/IP networks. IP packet is called datagram and the user data is a packet with the IP header. The normal size of the IP header is 20 bytes, unless options are present as shown in Fig. (2). With options and IPV6 information, the maximum size of IP header is 60 bytes [22].

More than encapsulate data to be sent, IP has 2 important features: MTU measurement and TTL.

MTU is the Maximum Transmission Unit of a link layer. A network normally has a upper limit of data that can be sent in a package. If IP has a datagram to send and it is larger than the MTU, it will be broken into fragments so that each fragment is smaller than MTU [24]. If a datagram has to be fragmented, it will have bigger overhead because each fragment will need a new IP header. IP is implemented independently of the link layer so, the MTU size can be of any size. Meanwhile, the ethernet payload size id adopted as a standard size and it is defined as 1500 bytes and thus defines the upper bound on segment size [13].

TTL is the Time to Live unit of time to a network packet. If a datagram is sent and its time to live exceed, the devices on the packet path can just ignore it and remove it from the sending queue. This feature is directly attached to the packet loss feature of an application and IP will do the better to send all packets.

C. User Datagram Protocol (UDP)

UDP is the most simple transport protocol that runs over IP. As the IP packet, UDP message is called datagram. UDP protocol will just encapsulate the application data with its 8 bytes header [24] (shown in Fig. 3). Thus, UDP is a lightweight transport protocol with a minimalist service model that does a little more work than to simply demultiplex messages to the application layer [12] [25].

As the normal IP datagram, UDP messages can be lost, duplicated or arrive out of order [21]. That is why it is called an unreliable transport protocol. The only guarantee that a UDP datagram has is the checksum. Each datagram is handled independently from all other datagrams. Checksum will try to grant that each datagram content is consistent. However, UDP doesn’t specify what will be done if the checksum doesn’t match since the protocol doesn’t predict package retransmission. Thus, applications that use UDP must be prepared to deal with error recovery, checksum failure, packet loss, packet reordering, flow control, congestion control and so on [26] [22].

In the application level, each output operation by a process produces exactly one UDP datagram, which causes one or more IP datagrams to be sent, depending on the network physical MTU. Theoretically, the maximum size of an IP datagram is 65535 bytes to be used to data and headers [24]. Although many implementations will accept larger datagram, all implementations are required to accept IP datagrams of 576 bytes. It means that it will use 8 bytes to UDP header, 60 bytes to IP header (maximum-size) and 508 bytes to application data [22].

Finally, UDP has no handshaking or acknowledge messages, neither initial or final handshaking. There is no way to know if a packet was correctly sent or if a client is still connected to a server [12].

D. Transmission Control Protocol (TCP)

TCP is the classical reliable transport protocol from TCP/IP suite. Since TCP is a byte-stream protocol, the receiver will receive segments of the data stream. That is why the TCP message is called segment. Each segment is packaged with a 20 bytes TCP header as depicted in Fig. 4.
TCP provides a connection-oriented, reliable, byte stream service [24]. To provide reliability, TCP communication does the following:

1) **MSS**: Application writes data through TCP using the write function. To send a segment, TCP can accumulate data from several writes into one segment or split the data from one write over multiples segments. For this reason, the number of bytes read by the receiver may not be the same as the number written by a sender [25]. The largest chunk of data that TCP will send is called Maximum Segment Size (MSS). When a connection is established each side can inform its MSS [24]. This kind of data control tries to prevent that a sink node overflows a source with a data larger than it can carry. Every TCP communication must agree with the amount of data that will be sent.

2) **ACK**: When TCP receives data it sends an acknowledgment packet (ACK). It informs sender that the segment was correctly received.

3) **ACK timeout**: TCP maintains a timer to each segment sent, waiting for the other end to acknowledge the reception. If an ACK is not received in time the segment will be retransmitted.

4) **Checksum**: As UDP, TCP has its checksum header field to check if message is corrupt or not. TCP will discard a segment with invalid checksum and will not ACK receiving it. It expects the sender to retransmit it when the ACK timeout expires.

5) **Buffer**: Independently of MSS, TCP will be carried by IP datagram and must fit in the IP payload [13]. Since IP datagram can arrive out-of-order, a TCP receiver has to sort the data if necessary. The packet sorting is done by the Transport Layer and every data received by application will be in order. The sequence number into TCP header is used to do this. Sequence number is used also to discard possible duplicate data no matter if the duplication occurs because a IP datagram duplication or an erroneous resent segment.

6) **Flow control**: Each end of TCP connection has a finite buffer space. Thus, a receiver informs a TCP sender how much buffer space it has to receive data, preventing overrunning its capacity.

7) **Congestion control**: While flow control is an end-to-end agreement, congestion control cares about the link capacity. This control will prevent TCP inject more data than the link or switches can stand [25].

As a connection oriented protocol, TCP uses a three-way handshake to establish a connection. The three-way handshake is completely transparent to the client and server programs [12].

Due to the reliability of TCP, its connection is always point-to-point. A multicasting communication that sends data to many receivers is not possible in TCP [12].

### E. Stream Control Transmission Protocol (SCTP)

SCTP is a connection-oriented transport protocol that provides a reliable full-duplex association [23]. This protocol was originally not meant as an replacement to TCP, but developed to transport voice over IP (VoIP) [27]. The SCTP packet contains a fixed 12 bytes header more a 4 bytes header to each chunk, as depicted in Fig. 5. A chunk is the unit of information within a SCTP packet. Each chunk contains its own header with the chunk type, chunk flags and chunk length [23].

The reliability of SCTP is provided with the following features:

1) **Multihoming**: SCTP provides associations between clients and servers. The word “association” is used instead of “connection” because it can involve more than two addresses due to multihoming. This feature can provide increased robustness against network failure. An endpoint can have multiple redundant network connections, where each of these networks has a different connection to the Internet infrastructure. SCTP can work...
around a failure of one network or path across the Internet by switching to another address defined in the SCTP association [23].

2) **Multiple streams:** SCTP can provide multiple streams between connection endpoints, each with its own reliable sequenced delivery messages. If one message is lost in one of the streams, SCTP does not block the delivery of messages in any of the other streams [23].

3) **Transport-layer fragmentation:** SCTP keeps a fragmentation point based on the smallest path MTU found to all the peer’s addresses and this smallest MTU size is used to split large user messages into smaller pieces that can be sent in one IP datagram [23]. Like TCP and unlike UDP, the fragmentation occurs at the transport-layer instead of the IP layer.

4) **Fast retransmit:** SCTP, like TCP, uses ACK to inform a packet loss. But instead of a normal ACK, it uses a selective acknowledgement (SACK) and a mechanism that sends SACK messages faster than normal when losses are detected [22].

5) **Non-blocking head of the line:** Unlike TCP which is stream-based, SCTP is message-oriented or message-based like UDP. This allows SCTP to separate different signaling messages at the transport layer. SCTP can deliver signaling messages to the application as soon as they arrive [22].

SCTP, like TCP, also provides reliability, sequencing, flow control, congestion control, and full-duplex data transfer through message chunk building, checksumming, packet validation and path management [23].

There are two types of SCTP sockets implementation: an one-to-one socket and an one-to-many socket. An one-to-one socket corresponds to exactly one SCTP association.

During the initialization process, SCTP uses a 4-way handshake instead of a 3-way used by TCP [28].

**F. Datagram Congestion Control Protocol (DCCP)**

DCCP is a transport protocol that combines TCP-friendly congestion control with unreliable datagram semantics for applications that transfer fairly large amounts of data [29] [17]. It uses a minimal 12 bytes header as shown in Fig. 6 to avoid the network overhead [30].

DCCP may use some ACK and does not understand a checksum error as a network congestion problem. As DCCP does not use retransmissions, TCP fast-recovery mechanisms are not implemented [32]. It is developed to delay-sensitive applications that prefers timeliness to reliability. Unlike UDP, that doesn’t provide this feature, DCCP will avoid congestion collapse. It aims to add to UDP a minimum mechanism to support congestion control, such as reliable transmission of ACK information [31].

DCCP, like TCP, provides a single bidirectional unicast connection: both data and acknowledgements can flow in both directions. At the beginning of the connection the endpoints must agree on a set of parameters such as the congestion control mechanism to be used [31].

**G. Comparison Draft**

To easily show the main differences among these protocols, a tabular view is depicted bellow.

<table>
<thead>
<tr>
<th>Feature</th>
<th>UDP</th>
<th>TCP</th>
<th>SCTP</th>
<th>DCCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message oriented</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Connection oriented</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Full duplex</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Reliable data transfer</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Ordered data delivery</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Unordered data delivery</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Flow control</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Congestion control</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multicasting</td>
<td>Yes</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Path MTU discover</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Checksum</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Header size (in bytes)</td>
<td>12</td>
<td>20</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

III. IMPLEMENTATION IN PURE DATA

The implementation of an application that can distribute audio throw network was made using Pure Data\(^2\), a real-time graphical programming environment for audio, video, and graphical processing. Pure Data allows the development of externals, a kind of plug-in, in C programming language. This externals can be attached to the application core and use its default externals to build a sound application (called patch). A Pure Data external library was developed with 5 network implementations: UDP, TCP, SCTP one-to-one, SCTP one-to-many and DCCP.

The external was developed in C using the following strategies: TCP was tuned with NO\_WAIT flag to avoid packet loss and have less latency; the multichannel audio data were concatenated as one single data to be send but in SCTP that allows multichannel; although all protocols being full-duplex, server always sent data and client always received it; server sent data as soon it is performed by Pure Data; client used a ring buffer to store received data after send it to Pure Data perform it.

The sender object, presented in Fig. 7, has to inform how many audio channels will be available and which protocol

\(^2\)http://puredata.info/
will be used. The input audio will be plotted to allow a visual comparison.

![Figure 7. Pure Data patch with medusa receive](image)

To easily compare the result, a loopback client was installed in another machine. A loopback just send back the data received to the sender, as depicted in Fig. 8.

![Figure 8. Pure Data patch with Medusa Loop-back](image)

The receiver, as the sender, should specify the protocol and the amount of channels. Different of the sender, a receiver should also inform the client IP to be connected.

![Figure 9. Pure Data patch with medusa receive](image)

In the sender machine, a receiver will get the loopback audio to plot and compare the results. The result is presented in Fig. 10.

![Figure 10. The plotting of the result](image)

In this plot is possible to see the phase difference between the sent and the received sinusoid. This difference is the latency of the network influencing the transported audio.

**IV. Conclusion**

UDP is a simple, unreliable, connectionless protocol, while TCP is a complex, reliable, connection-oriented protocol. SCTP combines some of the features of both protocols, providing additional features beyond those found in TCP. DCCP is as simple as UDP but brings the congestion control to the datagram communication. All these protocols can be used in audio communication and the present paper introduced a implementation as a Pure Data external. The result achieved is not exactly a test but a visual and sonic confirmation that the communication involving different protocols can bring new fields of research in network music context.

**A. Future Works**

Real results of these protocols comparison should be achieved. The development of a medusa_meter was started to try to compare bandwidth, latency and packet loss using the same scenario: a loopback station and the possibility of compare the sent data with the received one.

**V. Acknowledgment**

The authors would like to thank the support of the funding agencies CNPq (grant n° 141730/2010-2) and FAPESP - São Paulo Research Foundation (grant 2008/08623-8).

**References**


