Continuous Real Time Data Transfer with UDP/IP

1Emil Farkas and 2Iuliu Szekely

1Wiener Strasse 27 Leopoldsdorf I. M., A-2285, Austria, farkas_emil@yahoo.com

2Transilvania University of Brasov, Eroilor 29, 500029, Brasov, Romania, szekelyi@vega.unitbv.ro

Abstract

The number of practical applications when data from remote locations has to be transferred in near real time and continuous mode to a data acquisition center using the IP protocol over the Internet or a private network is increasing. The present document discusses the advantages of using UDP/IP versus using TCP/IP for these networks and is trying to provide solutions to the most important issues. Discussions and solutions in this paper are based on practical observations.

Keywords

Internet, Continuous data transfer, UDP/IP Protocol, Error correction

1 Introduction

The Internet Protocol (IP) became the most widely used protocol for data communication over most network types. It is the workhorse of data communication on the Network layer. One layer above, on the Transport layer there are two protocols relying on the services of the underlying IP: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), [1], [2], [3], [4].

Most of the applications use TCP/IP, while only a few applications rely on UDP/IP for the delivery of data, from reasons, which become evident below. However, there is segment of communications applications in which UDP/IP could be very successfully used for data transfer. This is the field of large data acquisition networks, more precisely, networks built to transfer near-real-time data from a remote location – e.g.: a measuring or monitoring station – to a data acquisition center [6]. Data transfer is a constant process and is required to be continuous, with no gaps and interruptions. Communication between the sender equipment – typically a digitizer - and the receiving equipment – typically a data acquisition computer - is completely stateless. When you turn on the sender, it sends data unsolicited. UDP/IP can be used in these networks with high efficiency, provided that the sending and receiving applications are implemented to fill-in for the deficiencies of UDP/IP concerning reliability of data delivery to the destination. That is overlaying applications have the tasks of data recovery and flow control.
2 Problems with Using TCP in Continuous, Real-time Data Transfer

Even though TCP and UDP both use IP as protocol on the network layer, TCP provides connection oriented, reliable byte stream service, while UDP is just a simple, datagram oriented, transport layer protocol [2]. TCP over IP achieves end-to-end reliable transmission of data across a network [1]. TCP flow control uses a sliding window flow-control protocol with variable window size [1].

Using TCP/IP, the sender starts with a window size equal to that of one TCP segment, [1]. The receiver acknowledges the IP datagram delivered to the destination workstation [1]. The sender then increases the window size to two segments and if it receives acknowledgements for both segments it increases the window size to three segments [1]. It continues to increase the window size in this way until a point when the link to the receiver becomes congested and acknowledgements to one or more segments are not received within a certain time-out, after what the sender re-transmits the segments to which acknowledgements were not received [1]. Upon such re-transmission the sender decreases the window by one segment [1]. Using this dynamic window flow control protocol, TCP continuously “probes” the link for its available bandwidth and, based on this information dynamically adjusts the speed with which it “pumps” data into the link.

However TCP has a number of disadvantages that in turn result in low efficiency of TCP when used for continuous data transfer in various scenarios. Some of these characteristics are:

- **Relatively large overhead on the inbound link**: The TCP header is constructed with 20 bytes – or more if the Options fields are used [2]. In case of TCP the amount of data written by an application may have little relationship to what actually gets sent in a single datagram. One data packet generated by the overlaying application could be segmented into several datagrams, each requiring one header, thus increasing the overhead.
- **Overhead on the outbound link**: When using TCP, the outbound link is loaded with the acknowledgements generated by the receiver.
- **Continuity of data transfer**: TCP has proven very inefficient when used over long-delay networks [6], such as satellite links, especially when Time Division Multiple Access (TDMA) schemes are used. Even when there is plenty of bandwidth for the transmission of respective data on the long-delay network, TCP will have to wait for acknowledgment of the sent segment before sending the next one. If this acknowledgment comes after the sender’s time-out, the respective segment is unnecessarily re-transmitted, causing a delay in the continuous data transfer.
- **Lack of ability for multitasking.** If wanted to send the data to a set of hosts using TCP, a separate connection has to be established between the sender and each recipient. If portions of the link are common, this translates into having to send each packet as many times as many destination hosts are.

Below are presented a number of graphs taken on data transfer using TCP/IP in continuous data transfer. The purpose is to demonstrate the deficiency of TCP/IP in certain scenarios when used in continuous data transfer. In the graphs from below the horizontal axis represent...
packet origin time while the vertical axis packet receive time. For start and comparison, the graph in figure 1 presents a perfect data recovery after a temporary link outage, typical for situations when adequate bandwidth is available and the link is not a delayed link.

**Figure 1 - Correct data recovery in FIFO (first-in-first-out) order**

The graph in figure 2 presents a typical scenario when data packets are received out-of-sequence. This is usually caused when TCP acknowledgements are received in a slower pace than data packets are generated – it should be noted here, that in most of the continuous data acquisition application the sender is required to always assign the highest priority to real-time packets. In this case when acknowledgements are received the sender will skip the packets created while was waiting for the acknowledgement and will send the most recent packet. Skipped packets will be put in the transmit queue inserted in between real-time packets.

**Figure 2 - Out-of-sequence packet reception**
3 Advantages of Using UDP for Continuous Real-time Data Transfer

In contrary with TCP, UDP provides no reliability: it sends the datagram that the application writes to the IP layer, but there is no guarantee that they ever reach their destination [2]. UDP does not have the mechanism either to probe the link for the available bandwidth, therefore it simply “pumps” as much data into the link as much is written by the application to the IP stack and if this is more than what the bandwidth of the link would allow, than datagrams are dropped on the link, without the knowledge of the sender.

The connectionless characteristic of UDP would simply imply that applications should be designed for using TCP, while completely avoiding UDP. However, UDP is useful in situations where the reliability mechanisms of TCP are not necessary, such as in cases where a higher-layer protocol provides error and flow control [4]. UDP is the transport protocol for several well-known application-layer protocols, including Network File System (NFS), Simple Network Management Protocol (SNMP), Domain Name System (DNS), and Trivial File Transfer Protocol (TFTP) [3]. UDP has some major advantages in comparison with TCP for use with continuous data transmission:

- **Less overhead:** While the TCP header is constructed with 20 bytes – or more if the Options fields are used -, the UDP header is only 8 bytes. [2].
- **Simple encapsulation on the IP stack:** Each output operation of the respective application using the services of UDP produces exactly one datagram, thus only one IP datagram is sent [2]. Considering that in case of TCP the amount of data written by an application may have little relationship to what actually gets sent in a single datagram, the same amount of data requires one 8-byte header when sent over UDP [1].
- **Continuity of data transfer:** UDP simply forwards the data on the link, as soon as these are received from the application, without waiting from receiver’s acknowledgements. This allows for a continuously flowing data transfer.
- **Capability of UDP for multicasting:** Multicasting is very often required in continuous data acquisition to allow an application to send the data to multiple recipients. Multicasting, like broadcasting, only applies to UDP [2].

Considering all the advantages, listed above, of UDP versus TCP, UDP could be very efficiently used for continuous data transfer. The only problem remains to be solved is the lack of flow control and error correction of UDP. These roles should and could be passed to the overlaying application.

4 Error Correction and Flow Control in Continuous Data Transfer with UDP

Error correction consists of error detection and data recovery. Error detection can be easily implemented by simply turning on the use of UDP checksums [1], [2], [3], [4]. While checksum with TCP is mandatory, with UDP is only optional and it can be turned off and on [2]. However, in fact it is strongly recommended turning on the UDP checksums if data is leaving the LAN (Local Area Network) [2]. This is because errors caused by software or hardware bugs of routers, bridges and other network interconnection devices can modify bits
of the datagrams [2]. These modifications are undetectable in a UDP datagram if the end-to-end checksum is disabled [2].

The UDP checksum is calculated both on the header and the data sections of the datagram and if the checksum check fails at the receiving end the entire datagram is simply discarded. However, UDP does not have the mechanism required for the recovery, from the sender (re-transmission), of discarded datagrams. Therefore this role is left to the overlaying applications [4]. Data recovery could be done either through positive or through negative acknowledgments [6]. Positive acknowledgements are sent by the receiver to the sender for each data packet received, while negative acknowledgements are nothing else than requests for re-transmission of discarded data packets [6]. This latter solution uses less bandwidth on the outbound channel, since acknowledgements are sent only when packets are discarded and not for every single one. However, use of negative acknowledgments requires the identification of each individual packet with a unique number. The simplest is to assign a sequence number to each packet, thus allowing the recipient to realize when a number is skipped at reception. The sequence number in our example presented in fig. 3 is an unsigned 3-byte integer.

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Version</td>
<td>Header length</td>
<td>Type of service</td>
<td>Total length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identification</td>
<td>Flags</td>
<td>13 bit fragment offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TTL</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source IP address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination IP address</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Options</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source port</td>
<td>Destination port</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UDP length</td>
<td>UDP checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Header</th>
<th>Data Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source ID</td>
<td></td>
</tr>
<tr>
<td>Packet sequence number</td>
<td></td>
</tr>
<tr>
<td>Oldest packet sequence number</td>
<td></td>
</tr>
<tr>
<td>Time stamp (long seconds)</td>
<td></td>
</tr>
<tr>
<td>Time stamp (long seconds)</td>
<td></td>
</tr>
<tr>
<td>First sample value</td>
<td></td>
</tr>
<tr>
<td>Data (differences)</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3 - Structure of a data packet supporting negative-acknowledgment-based (re-transmission requests) data recovery, encapsulated in a UDP/IP datagram.**
Fig. 3 presents an example of a packet format encapsulated in a UDP, then in an IP datagram, suitable for the implementation of data recovery through re-transmission requests (negative acknowledgements). Data are compressed using a first-difference compression algorithm.

Re-transmission request for a packet is generated when instead of the packet with expected sequence number a packet with a greater sequence number is received. After the receiver detects a sequence number higher than the one of the expected packet, it generates and sends to the sender requests for re-transmission of the packets not received. There are a number of issues to be addressed when implementing a negative acknowledgement data recovery scheme:

- **Data buffer at the sender.** The very minimum requirement for data recovery is data buffering capability of the sender. The sender should be able to keep in its memory all of the transmitted packets for time greater than the time required by the sender to receive and process the received packet, as well as to generate a re-transmission request. This memory should be of a circular type based on FIFO (first-in-first-out) order, packets being identified by their sequence number [6].
- **Type of protocol for re-transmission requests.** Re-transmission requests could be transferred from the receiver to the sender on either UDP/IP or TCP/IP. TCP/IP is more recommended, since this ensures reliable delivery.
- **Destination address of the re-transmission request.** The header of the inbound packet contains a unique identification number (data source ID) of the host generating the data packet and the IP address of the data packet. Although the IP address would be enough and adequate in proper addressing of the re-transmission request, since the identification number is already included in the data packet for other data processing purposes, this could also be used when addressing the request. The destination IP address is useful only when the request is ported by IP all way to the destination. However, it is „lost” as soon as the request-packet is forwarded onto another type of network. Therefore the use of the ID in the address field of the request proves to be very useful when the request is required to leave the IP protocol and transfer over a bridge onto a different type network not using IP.
- **When to send the re-transmission request after detecting a missed packet?** Although is not common, packets of the same data stream could arrive to the sender on different routes, therefore arriving to the receiver out of sequence. When low amount of data are being transferred from the sender to the receiver – typical to most of large-area acquisition networks – packets arrive to sender most of the time in sequence. The amount of re-transmission requests generated for out-of-sequence packets is low and does not pose a risk of overload the communication link. In case of higher (in range of MegaBytes) data rates – and these are the exceptional cases – the receiver could generate and send the re-transmission request for a missed packet after a certain time-out, thus giving more chance for the missed packet to arrive. This would cover the cases when packets take different routes.
- **How many re-transmission requests to be generated at a time?** In situations when an entire range of data is missed – due to communication outage – the receiver should generate one single re-transmission request for the entire gap in the data segment. This would avoid overloading the outbound link.
- **How many re-transmission requests to put in the outbound transmit queue at a time?** There are situations when re-transmission requests have to be generated for filling in many short consecutive gaps. In these cases the sender will be expected to reply to a
relatively high re-transmission requests in a very short time, thus overloading the inbound link. This would generate further gaps at the receiver and could initiate an avalanche of gaps from which the system could never recover. The best solution to this problem is to limit the number of re-transmission requests the receiver could send to the sender. This number would then constitute the length of a sliding window: once data is received for one re-transmission request the sender would issue the next-in-sequence request. This solution slows down the pace of missed data recovery, but the system eventually recovers all data and inbound link is not overloaded over a limit from which the system would not recover.

- **What if the packet is not received after being requested?** The receiver can continuously check which packets have been received – using the packet sequence number for identification of packets – and could repeat the request if the already requested packet has not yet been received. The length of this repeat interval could be user configurable and it could depend on the speed of the inbound link (the slower the link the longer the interval). The sequence number of the oldest packet available in the sender’s memory is included with every inbound packet, therefore the receiver will know when to consider a packet definitely lost and stop repeating the requests for the re-transmission of this packet.

All of the above would materialize in a number of parameters used for tuning the system and the communication. These parameters could either be hard-coded, or user configurable or changed dynamically by the application based on reception statistics. Hard-coding these parameters would result in a very rigid system lacking the customization ability to communication links with different performance. However, allowing the application to dynamically adjust these parameters based on reception statistics would decrease the manageability of the system. Therefore, the most feasible solution is if these parameters are user configurable.

TCP is working on a positive acknowledgement scheme [1], [2], [3], [4], [5]. This means that the sender interrupts data transmission until a receipt acknowledgement is received from the receiver for the data segments sent in a certain size window. It can be said that TCP probes the link for available bandwidth and adjusts the flow of data accordingly. Because UDP/IP lacks all of this mechanism, the role of flow control should be implemented in the overlaying protocol, on the application layer [2]. To preserve the advantages of UDP, the scheme to be implemented should be one based on the negative acknowledgement scheme described. The sender, when using UDP/IP does not have any information or possibilities to learn the real throughput of the link at any given time. The receiver should assume this role. At startup and when is not instructed otherwise the sender transmits with the packet generation rate. It would slow this pace, however, if it receives special instructions in this regard from the sender. The receiver can probe the link from the sender using some or all of the following information:

- **The number of missed packets.** The receiver can simply evaluate, to a certain precision, the performance of the link by monitoring the number of packets missed in a certain pre-defined time interval and send a message to the sender to “slow down”. The parameters, which control the pace, with which the sender puts packets in the transmit queue are: the number of packets sent in a time period and the time interval between consecutive packets.

- **The density of data gaps at the receiver.** If the gaps in the received data become denser the receiver can instruct the sender to decrease the pace of transmission.
• **The length of data gaps at the receiver.** If gaps in the received data are short and dense, than these are caused by low link performance. However, long gaps are probably due to some other reasons, for instance, total link outage. In this case the receiver does not instruct the sender to “slow down”.

In fact the flow control described above is very similar to the one used by the TCP/IP. The major difference is that the sender does not wait for receipt acknowledgements. In summary:

a. The sender starts sending data as soon as it is started and continuous to do this with the maximum rate at which data is being created. It is a stateless, continuous transmission.
b. The sender re-transmits any packet stored in its memory if a request is received for this from the receiver.
c. The sender decreases and respectively increases the pace of transmission when instructed so by the receiver.
d. The receiver is ready to start receiving data from the sender or senders to which it was configured and continues to receive these packets as long as they arrive.
e. The receiver generates a re-transmission request when it realizes that a packet was missed.
f. The receiver, instructs the sender to decrease the rate of transmission if discovers gaps in the received data and, respectively instructs the sender to increase the transmission rate or resume normal operation if these gaps disappear.

5 Conclusions

As conclusions it can be said that in networks where data transfer from a remote data source to the acquisition center is required to be in near-real time, that is continuous, suing UDP/IP as the transport protocol has obvious advantages compared to TCP/IP. However, the full success depends profoundly from the overlaying application. The authors have presented one solution for the implementation of an application layer protocol that would assume the role of error correction when UDP/IP is used.

References