

Topic of the Month – October 2006

The Effect of Packet Jitter on Application Latency

When implementing real-time services over IP, an acceptable latency for the application to function properly must be determined in advance and met by the network infrastructure.

For this month's topic, we will examine the effects of packet jitter on the overall latency of an application. We will see that, in certain circumstances, packet jitter can have a significant impact on application performance, even when the average transfer delay is relatively low. Understanding the effects of jitter and compensating for it is especially critical in certain applications, for example when interconnecting cryptographic devices that require synchronization.

To simplify this discussion, we will assume a real-time service that requires transport of a constant bit rate data stream - a 500 kbps circuit emulated over an IP network. In this simplified case, the end-to-end latency can be divided into three distinct components: the packetization delay, the packet transfer delay, and the reassembly delay. We will ignore the insignificant processing delays on each end (for example on the receive side, that would be passing the packet up the protocol stack and notifying the application that the packet is available).

Throughout the discussion we will use the analogy of transporting water from one office to another. Imagine a water cooler that creates a constant stream of water when you pull the tap. We will recreate that stream of water in another office by filling cups of water (packets of bits), carrying the cups to the second office and then pouring them into a bucket with a spigot that creates the same constant stream of water.

Definitions:

Packetization Delay – The time required for the application to fill an IP packet to a specified level. This is analogous to filling the cup in the first office. In all of the examples below, we will assume an IP packet payload size of 1000 bytes and a circuit rate of 500 kbps, so the packetization delay is the packet size divided by the rate at which it is filled:

$$PD = 1000 \text{ bytes} * 8 \text{ bits/byte} / 500,000 \text{ bits/s} = 16 \text{ ms}$$

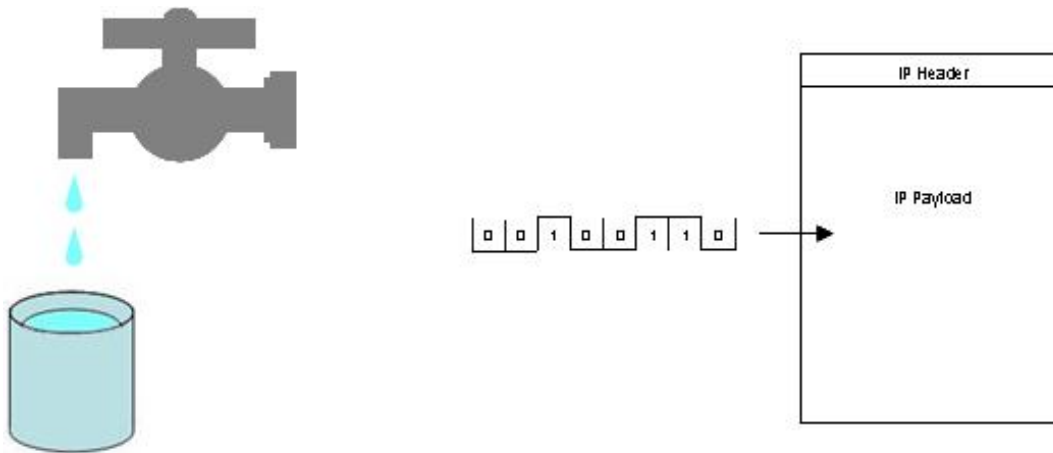


Figure 1 - Packetization Delay

Transfer Delay – Also, known as packet latency, this is the time elapsed from when the packet enters the IP cloud at the transmit end and when it exits the IP cloud on the receive end. In our analogy, this is the time required to deliver the cup from the first office to the second office.

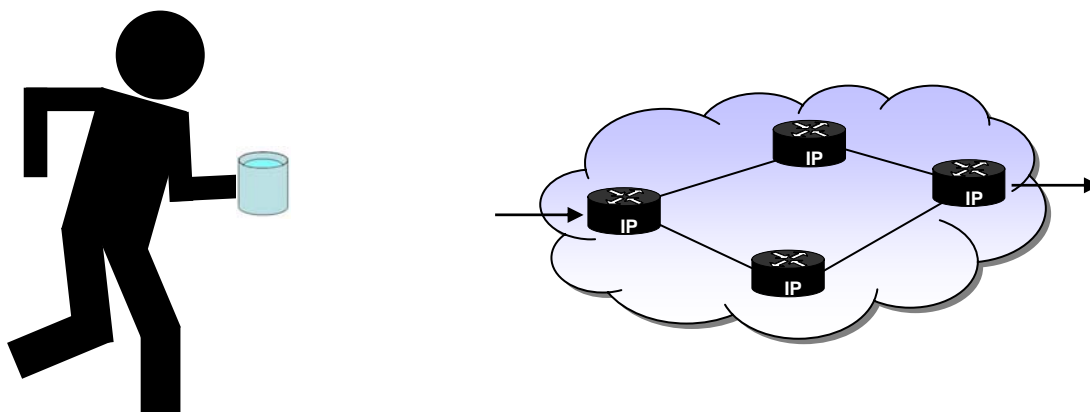


Figure 2 – Transfer Delay

Reassembly Delay – The amount of time the transported data waits in a buffer at the receive side before being read out by the application. In our analogy, this is the time required to fill the bucket to a level which ensures that your output stream is constant. When can you open the spigot and be sure that you have a constant stream of water coming out of the bucket? That depends on the arrival of the cups of water that you are pouring in. If you know you will always have a cup of water ready, you don't have to wait at all. If you are uncertain about the time the cup will be ready to pour, you should wait to open the spigot until you have enough water stored in the bucket to compensate for the delay in the arrival of the cup. The amount of time you wait to turn on the spigot is the reassembly delay.

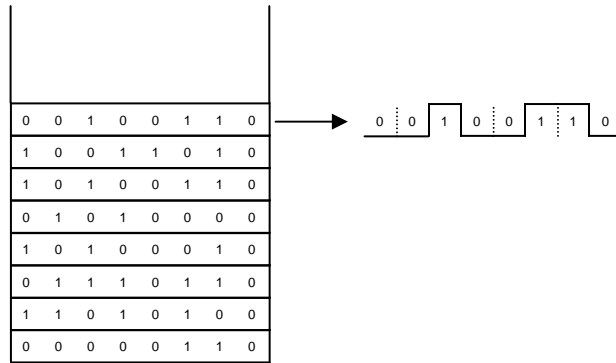


Figure 3 – Reassembly Delay

Application Latency – The elapsed time between when an application presents data to the transmit side and when the application reads out data on the receive side. In our case, we will ignore the insignificant processing delays on each end so the application latency becomes the sum of the packetization delay, the transfer delay and the reassembly delay. In our analogy, this is the delay between when water leaves the tap of the water cooler and when it leaves the spigot in the second office.

Packet Jitter – The variation in the transfer delay of IP packets. In our analogy, it is the variation in the time it takes to carry the cup from the first office to the second. It might take, on average, one minute to walk between offices, but different trips might take less time (if someone opens doors for you) or more time (if you have to wait for an elevator).

Example 1:

In this example, we will assume zero packet jitter. That is, the network delivers all IP packets from source to destination with exactly the same transfer delay every time (not a good real-life assumption but valuable to the discussion). We will assume that delay to be 40 ms.

We already know the packetization delay (PD) from earlier, it is 16 ms. We also know the transfer delay because we just assumed it to be 40 ms. So to determine the application latency, all that is left to calculate is the reassembly delay.

Remember that the reassembly delay is the time the data sits in a buffer at the receive side waiting to be read out to the application. The reason this buffer exists is to compensate for jitter (delay variation). Basically receive packets are stored until enough data is ready for the application so that the buffer never becomes empty. In this example, we have assumed that packets arrive at a constant rate: once every 16 ms (40 ms after it was transmitted into the IP network). In this case, we know that the next packet will be ready and waiting to read to the application when the buffer empties, so theoretically, we don't need to wait (and don't even need a buffer really) so the reassembly delay is zero.

$$\text{Application Latency} = 16 \text{ ms} + 40 \text{ ms} + 0 = 56 \text{ ms}$$

Example 2:

In this example, we will introduce packet jitter into the network. We will assume that the transfer delay is $40 \text{ ms} \pm 10 \text{ ms}$. To return to the water analogy, this means that the cups you would pour into the bucket arrive in the second office on average in 40 ms, but they could arrive early or up to 10 ms late. In a worst case scenario, the first cup you received could have taken only 30 ms to arrive and the next cup may take up to 50 ms. To ensure a constant stream at the output, you need to wait 20 ms after pouring the first cup in.

$$\text{Application Latency} = 16 \text{ ms} + 40 \text{ ms} + 20 \text{ ms} = 76 \text{ ms}$$

Example 3:

Now let's look at a real-world example. We performed a ping to a random site on the Internet and recorded the delay of 100 round trips (we realize that we need data for a one-way trip in this example, but we'll ignore that for simplification). The average delay was 54.4 ms, the minimum delay was 50 ms and the maximum delay was 132 ms. Here's a chart of the data:

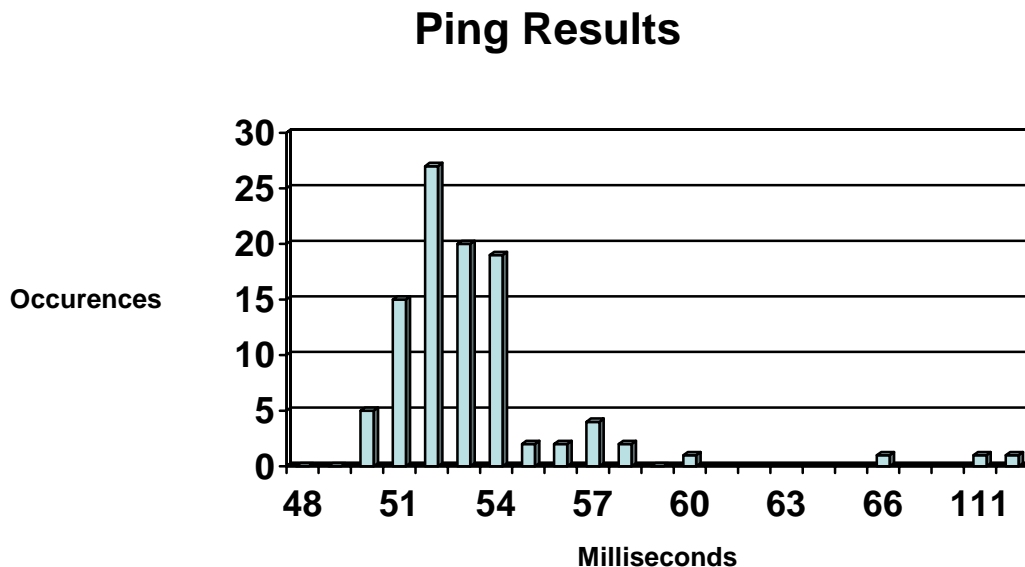


Figure 4 – Ping Results

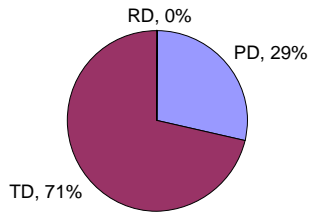
The maximum delay variation in these results is $132 \text{ ms} - 50 \text{ ms} = 82 \text{ ms}$. This is also the time we need to wait to read out the initial data to the application, or the reassembly delay. So the overall application latency is:

$$\text{Application Latency} = 16 \text{ ms} + 54.4 \text{ ms} + 82 \text{ ms} = 152 \text{ ms}$$

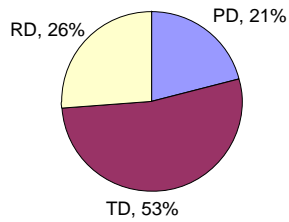
Results:

Now let's take a quick look at how the differing amounts of packet jitter affect the overall application latency.

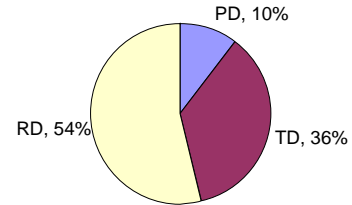
Percentage of Delays
Example 1



Percentage of Delays
Example 2



Percentage of Delays
Example 3



■ PD = Packetization Delay
■ TD = Transfer Delay
■ RD = Reassembly Delay

The obvious conclusion is that an increase in packet jitter causes an increase in overall application latency due to the need to buffer data on the receive side. The greater the jitter, the greater the reassembly delay. In certain cases (for example a two-way conversation), it may be impossible to implement the service if jitter is not controlled.

There are several methods to control the amount of jitter in the network such as IP DiffServ, MPLS, 802.1p, or ATM QoS. Each of these methods uses buffering schemes that are more advanced than the standard first-in-first-out (FIFO) buffering that is typical of best-effort IP networks. We will take a closer look at buffering techniques in a future Topic of the Month.