Universiteit van Amsterdam

SNE
System and Network Engineering

Time Sensitive Application Transport

Mohammad Shafahi *mohammad.shafahi@os3.nl*
Supervisor:
Cees de Laat *delaat@uva.nl*

July, 2012

Abstract

Today's most demanding applications are the ones that are time sensitive. This demand has introduced discussions in the Networking community, specially the NREN community, as to which technology best suites time sensitive application transport. In this research we have contributed to the effort of finding the suitable technologies by providing a framework for discussion and comparison between technologies. We have also suggested a software tool for doing so, although having in mind that hardware solutions might give more precision.

Contents

List of Figures

List of Tables

1 Introduction

Today's most demanding applications, known as time sensitive applications, are the ones that are sensitive to when data is delivered to them. Time sensitive applications are used in a range of different environments. For example voice or video calls are used in public while virtual machine migration and high speed stock market transactions are used in the industry. These application are also used in the scientific community, for example they are present in high performance computing clusters and live correlation done with multiple streams of data received from radio telescopes to improve resolution. They are even known in environments like medicine such as telemedicine and online surgeries.

Although all these applications are time-sensitive, the quality they require for this sensitivity varies. The quality required by applications can be classified into two main categories, the magnitude of latency and variation of delay (jitter).

Applications like voice or video calls require delays of less than 100 ms (see [9]) but are sensitive to jitter, while other applications like high speed transactions require delays as low as 2-3 ms (although will work with higher delays) but are not very sensitive to jitter (see [7]).

This growth of time sensitive applications has introduced discussions in the networking community specially the national research and education network's (NREN) community as to which technologies best suite time sensitive data transfer.

In this research we have tried to contributed to the networking communities effort of finding suitable technologies for time sensitive applications by proving a framework for discussion on the subject.

1.1 Research Question

The main research question for our research is "which technologies best suite time-sensitive data transfer?" To answer this main question we will need to answer these sub questions:

- What are the sources of delay and jitter in the technologies?
- What are the magnitudes of each source of delay and jitter?
- What are the requirement for measuring delay and jitter for the technologies?
- What are the suitable methods and tools to measure delay and jitter for detecting the best suited time-sensitive technology?

2 Sources of delay and jitter

Delay Delay or more precisely one-way-delay is the time it takes for a packet of data to travel from one node to another. There are many sources that cause delay in a network, thus for the sake of simplicity sources of delay in networking are categorized into four categories. These categories are propagation delay, processing delay, queuing delay and transmission delay.

Figure 1 shows these categories and where they are positioned in an intermediate network device (or hop). In this research we will not consider end nodes as part of the network but we should note that an end node device will only have half of figure 1 (The sender will have the right half and the receiver node the left half).

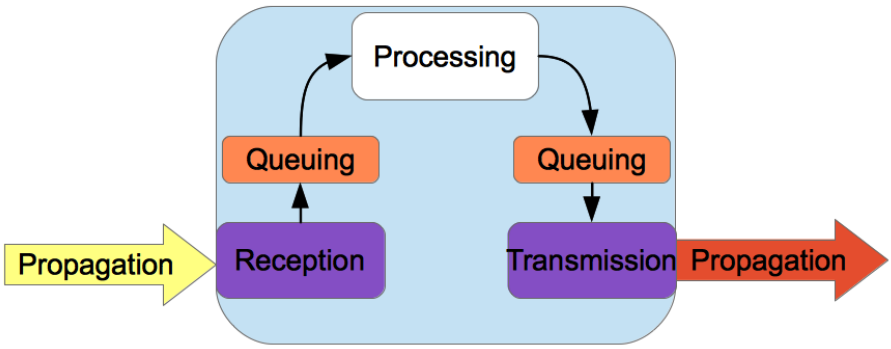

Figure 1: Sources of delay

Based on figure 1 and the fact that we don't consider the delay introduced by end users as network delay, we can define network delay based on formula (12)

$$delay_t = \sum_{i=1}^{N}(RD_t^i + QD1_t^i + PCD_t^i + QD2_t^i + TD_t^i + PPD_t^i) - PPD_t^N \qquad (1)$$

where PPD_t^i, PCD_t^i, RD_t^i, TD_t^i, $QD1_t^i$ and $QD2_t^i$ are the propagation delay, processing delay, reception delay, transmission delay, first queuing delay and second queuing delay for packet t at hop i.

We will talk about the details of each one of these categories in more detail later on. But to be able to continue we should define another concept, Jitter.

Jitter Base on [6] jitter (known by the RFC as delay variation) is defined as the difference between the one-way-delay of selected packets. This definition of Jitter is not precise and could be interpreted in different ways. To speak more precise we should have a formal definition of jitter. Jitter can be formulated as (2).

$$Jitter(\delta t) = Max(delays) - Min(delays) \tag{2}$$

where δt is a given time slot.

But these are problems in (2). In (2) you get a notion of how much the size of delay has changed but you don't know how this change is distributed in time so you might have just been unlucky once and have got a high delay for one of your packets but you have never got this again. To fix this you could use the variance of delay as defined in (3).

$$Jitter(\delta t) = \sum_{i=1}^{N} p_i.(d_i - \mu)^2 \tag{3}$$

where μ is

$$\mu = \sum_{i=1}^{N} p_i.di \tag{4}$$

in which δt is a given time slot, N is the number of different delays that happened in time slot δt and d_i is the amount of the ith delay and p_i is the probability of the ith delay in that time slot.

This formalization has it's own issues also, first of all it is not in the same order as $time$ but it is in order of $time^2$. To fix this we could take the square root of it and define Jitter as (5).

$$Jitter(\delta t) = \sqrt{\sum_{i=1}^{N} p_i.(d_i - \mu)^2} \tag{5}$$

Although (5) looks like a good definition for Jitter, it is only the standard deviation of delay. In this research we use a different definition for Jitter based on [1]. In this definition we calculate Jitter for each arriving packet (Except the first one) using equation (6).

$$Jitter_i = delay_i - delay_{i-1} \tag{6}$$

where $jitter_i$ is the jitter for the ith packet arrived at the destination. And we have (7) for Jitter for time slice δt.

$$Jitter(\delta t) = \frac{\sum_{i=1}^{N} |jitter_i|}{N} \tag{7}$$

Note that the first packet is packet number zero. Now that we have precise definitions for jitter and delay we will talk about each source in more details.

2.1 Propagation delay

Propagation delay is introduced by the time it takes for the traveling agent, that contains the data, to travel through the medium it is traveling through. For example if we are using light waves as our traveling agent and if the medium is glass fiber, propagation delay is the time it takes for light to travel through glass for the desired distance. Propagation delay is calculated using (8).

$$PPD = \frac{length}{propogationspeed} \tag{8}$$

2.2 Processing delay

Processing delay is the time it takes for a packet to be processed by a network node. Processing packets can vary from only a partial header inspection to deep packet inspection and encryption. While header inspection normally takes under microseconds, deep packet inspection and encryption can take much more time. For this processing delay can be a source of jitter.

Normally firewall nodes are responsible for deep packet inspection and it is not done at each node. Later on we will show that deep packet inspection can have an important role in the unpredictability of jitter in a connection.

2.3 Queuing delay

Queuing delay is the time a packet has to wait before it can get service. In a network node we have two queuing delays, the first one is introduced when packets have to wait before they can be processed by the node. This happens when the node's processing unit is overloaded for some reason.

The second queuing delay is introduced when packets have to wait before they are transmitted because the transmission units full capacity is in use. This delay is also a source of jitter and can introduce unpredictable amounts of delay that vary from zero where no other packet is in the queue to infinity in case of the buffer-bloat effect (See [13]).

Queuing delay can be calculated using (9) by having the used memory of the queue and service speed.

$$QD = \frac{UsedMemory}{ServiceSpeed} \tag{9}$$

Note that in (12), for $QD1$, $ServiceSpeed$ is the processing bit rate and for $QD2$ is the transmission bit rate.

2.4 Transmission and Reception delay

Transmission and Reception delay are the time needed to send and receive a packet on and from the link. These delays are introduced in the network interface of the device. They consist of the time needed to serialize (or deserialize for reception), calculate check-sums and access the media (only for transmission) for each packet. These delays can be calculated using (10) by just having the network interface speed.

$$RD/TD = \frac{1}{Interface\ speed} \tag{10}$$

2.5 Overview

Based on the sources of delay introduced previously we can identify which technology introduces what kind(s) of delay(s). Table (1) shows what kind of delay is introduced by some network technologies.

Tech/Delay	Propagation	Transmission/ Reception	Processing	Queuing
TDM		√		
(D)WDM		√		
SONET/SDH		√		
OTN		√		
Openflow			√	
MPLS-TP			√	
PBB-TE			√	
Optical	√			
Diffserv				√

Table 1: Sources of delay diffrent technologies

Given the sources of delay for each technology, we have a framework to discuss and compare different technologies by having the magnitude of the sources of delay. Knowing the magnitude of the sources of delay will also help us have an understanding of how much accuracy is needed in measuring delay of a network.

Table (2) shows an estimation of the amount of delay introduced by each source

Source of Delay	Magnitude
Propagation	4.9 μs/km [a]
Transmission/Reception	0.01 μs/kb [b]
Processing	2 μs/kb [c]
Queuing	$\frac{UsedMemory}{ServiceSpeed}$

of delay.

Table 2: Magnitude of sources of delay

[a] For fiber optics (The speed of light is 300,000 kilometers per second in vacuum and light travels about 3/4 of this speed in fiber)

[b] For a 100 Gb/s connection

[c] Could be much less (see [10])

3 Requirements of time sensitive applications

3.1 Sensitivity to delay

As discussed in the introduction the sensitivity to delay varies a lot for time sensitive applications. Based on [9] voice conversations can handle delays up to 150 ms and video calls can handle delays up to 80 ms so we can conclude that having delay lower than 80 ms is ideal for human interactive applications. Having this amount of delay is not competitive for high speed transactions in the stock market.

Now the question is how far can we go to reduce delay? If we take a closer look at the sources of delay we will notice the largest amount of delay is introduced by propagation delay and this delay is dependent on parameters such as speed of light in the conductor (for optical networks) and the length of the path. So the only way to reduce the amount of this kind of delay is to move your site as close as possible to the destination site or to try to come up with a conductor that passed the light with the maximum speed (hollow fiber).

Because our study is on transport networks reducing the distance is not a solution for us (Although we can reduce distance my using fiber instead of satellite or direct sight instead of fiber etc. but this is out of the scope of our research) and because other sources of delay are negligible compared to propagation delay, studying delay will not be a good choice for choosing suitable technologies for time sensitive application transport.

3.2 Sensitivity to jitter

As discussed before studying delay will not help identifying the suitable technologies for time sensitive application transport. But what can have an important role is jitter.

Studies show that jitter has an important role in the perception of delay in human interaction. This sensitivity of jitter can get as small as 5 μs. Table (3) shows some results of amount of tolerance to jitter for some user interactive applications.(See [16])

Video	Audio	Application	jitter
$\surd$		Overlay image	240 ms
$\surd$	$\surd$	Lip synchronization	80 ms
$\surd$	$\surd$	Music Script	5 ms
	$\surd$	Dialog	120
	$\surd$	Background music	500 ms
	$\surd$	Stereo	5 μs

Table 3: Tolerance of jitter for diffrent applications

4 Method

To decide on the method we want to calculate delay and jitter for a given network, we first should have a clear definition of network delay and jitter (delay and jitter introduced by the network).

4.1 Network delay and jitter

Network delay for packet i can be defined as (11).

$$delay_i^n = T_i^R - T_i^S \tag{11}$$

where T_i^R is the time packet i was received and T_i^S is the time packet i was sent. and based on (11), delay for time slice δt can be defined as (12)

$$delay^n(\delta t) = \frac{\sum_{i=1}^{N} |delay_i^n|}{N} \tag{12}$$

Network jitter (jitter introduced by the network) for packet i is defined as (13).

$$jitter_i^n = delay_i^n - delay_{i-1}^n \tag{13}$$

and based on (13) jitter for time slice δt can be defined as (14)

$$jitter^n(\delta t) = \frac{\sum_{i=1}^{N} |jitter_i^n|}{N} \tag{14}$$

In order to do these calculations we have two main methods of measuring these times, One way delay measurement (OWD) and Round Trip Time measurement (RTT).

4.2 Round Trip Time measurement (RTT)

In RTT a packet is time stamp with T_i^S and sent from a client measurement device to a server device. Then the server device echos the packet back to the client as fast as possible. Finally the packet is received by the client and time stamped with T_i^R.

This measurement method has the advantage that only with having "echo servers" at different destinations measurement is possible from our source to there. This measurement has lots of down sides. First of all the delay it calculates is a round trip, not a one way, so it is actually measuring delay introduced by the network twice if both paths are the same (That might not be the case). One

would suggest dividing the results by two but this will not work because then we are assuming that the send delay and the respond delay are the same that is not always true.

With this measurement method we are also assuming that the packet travels the same path in both directions. Even if we could ensure this assumption an extra delay exists and that is the time needed for the "echo server" to receive the packet and echo it and so the formula of delay changes to (15)

$$delay_i^n = T_i^R - T_i^S - T_{echo} \tag{15}$$

4.3 One way delay measurement (OWD)

In OWD a packet is time stamped with T_i^S and is sent from a client measurement device to a server device. Then the server device time stamps the packet T_i^R. In this method the two ends have to share the time stamps to calculate the measurement. (The time stamp could might sometimes be transferred with the packet itself).

Although this method solves the problems of RTT but it introduces the problem of synchronized clocks. For the measurement to be accurate the clocks of the two sides should show exactly the same time, otherwise the difference between the times will also be counted as delay (may be added or subtracted from the actual delay). Synchronized clocks can be achieved with combining a GPS receiver and the NTP protocol (See [11]).

Note that synchronized clocks are needed for the measurement delay but are not needed for the measurement of jitter because jitter deals with difference in delay and a fixed amount added or subtracted from delay, is cancels out in the subtraction.

4.4 Accuracy of Measurement

In our discussion above we assumed that the time stamping of T_i^S and T_i^R are accurate but this is not true in reality. To be able to have an understanding of the precision of our measurement of delay and jitter of the network, we should have an estimation of how much the measurement devices are adding to the amount of jitter and delay.To do so we first define the observed delay for packet i as (16).

$$delay_i^o = delay_i^m + delay_i^n \tag{16}$$

where $delay_i^n$ is the delay introduced by the network for packet i (Network delay for packet i) and $delay_i^m$ is the delay introduced by the measurement device for packet i (Measurement Delay for packet i).

We already have definitions for network delay and jitter but we should now have definitions for measurement delay and jitter :

4.5 Measurement delay and jitter

Measurement delay for packet i can be defined as (17)

$$delay_i^m = T_i^S - T_i^{STS} + T_i^{RTS} - T_i^R \tag{17}$$

where T_i^{RTS} is the time registered as the received time of packet i and T_i^{STS} is the time registered as the sent time of packet i. Based on (17) the jitter introduced by the measurement device for packet i (Measurement jitter for packet i) is defined as (18)

$$jitter_i^m = delay_i^m - delay_{i-1}^m \tag{18}$$

and based on (13), measurement jitter for time slice δt can be defined as (19)

$$jitter^m(\delta t) = \frac{\sum_{i=1}^{N} |jitter_i^m|}{N} \tag{19}$$

Now that we have the definition of measurement delay and jitter, we can redefine observation delay and jitter based on the time stamps introduced.

4.6 Observation delay and jitter

Observation delay for packet i can be defined as (20).

$$delay_i^o = T_i^{RTS} - T_i^{STS} \tag{20}$$

and based on (20) the jitter observed for packet i (Observation jitter for packet i) is defined as (21).

$$jitter_i^o = delay_i^o - delay_{i-1}^o \tag{21}$$

and based on (21) jitter for time slice δt can be defined as (22)

$$jitter^o(\delta t) = \frac{\sum_{i=1}^{N} |jitter_i^o|}{N} \tag{22}$$

Now that we have clear definitions of the observation delay, network delay and messurment delay, let's see how they are related to each other.

Obviously we can show that observation delay for a given packet is the sum of network delay and measurement delay :

$$\begin{aligned} delay_i^m + delay_i^n &= (T_i^S - T_i^{STS} + T_i^{RTS} - T_i^R) + T_i^R - T_i^S \\ &= T_i^{RTS} - -T_i^{STS} \\ &= delay_i^o \end{aligned} \tag{23}$$

This also is true about jitter for a given packet:

$$\begin{aligned}
&jitter_i^m + jitter_i^n \\
&\quad = delay_i^m - delay_{i-1}^m + delay_i^n - delay_{i-1}^n \\
&\quad = (T_i^S - T_i^{STS} + T_i^{RTS} - T_i^R) - (T_{i-1}^S - T_{i-1}^{STS} + T_{i-1}^{RTS} - T_{i-1}^R) \\
&\qquad + (T_i^R - T_i^S) - (T_{i-1}^R - T_{i-1}^S) \\
&\quad = (T_i^{RTS} - T_i^{STS}) - (T_{i-1}^{RTS} - T_{i-1}^{STS}) \\
&\quad = delay_i^o - delay_{i-1}^o \\
&\quad = jitter_i^o
\end{aligned} \tag{24}$$

but let's see if $delay^o(\delta t)$ is equal to $delay^m(\delta t) + delay^n(\delta t)$.

$$\begin{aligned}
delay^m(\delta t) + delay^n(\delta t) &= \frac{\sum_{i=1}^{N} |delay_i^m|}{N} + \frac{\sum_{i=1}^{N} |delay_i^n|}{N} \\
&= \frac{\sum_{i=1}^{N} |delay_i^m| + |delay_i^n|}{N} \\
&\geq \frac{\sum_{i=1}^{N} |delay_i^o|}{N} \\
&\geq delay^o(\delta t)
\end{aligned} \tag{25}$$

So

$$delay^m(\delta t) + delay^n(\delta t) \geq delay^o(\delta t) \tag{26}$$

but because $delay_i^o$ is always a positive number the extreme happens and so $delay^o(\delta t)$ is equal to $delay^m(\delta t) + delay^n(\delta t)$.

Finally let's see if $jitter^m(\delta t)$ equal to $jitter^m(\delta t) + jitter^n(\delta t)$

$$\begin{aligned}
jitter^m(\delta t) + jitter^n(\delta t) &= \frac{\sum_{i=1}^{N} |jitter_i^m|}{N} + \frac{\sum_{i=1}^{N} |jitter_i^n|}{N} \\
&= \frac{\sum_{i=1}^{N} |jitter_i^m| + |jitter_i^n|}{N} \\
&\geq \frac{\sum_{i=1}^{N} |jitter_i^o|}{N} \\
&\geq jitter^o(\delta t)
\end{aligned} \tag{27}$$

so

$$jitter^m(\delta t) + jitter^n(\delta t) \geq jitter^o(\delta t) \tag{28}$$

but $jitter_i^o$ can be negative or positive so this is all we can say.

4.7 Our method for Time Sensitive Application Transport

Based on the fact that in transport networks send and receive delay are different not just because of different paths but also because of traffic policies (like QOS) we should use OWD for the method for measuring delay.

Also based on the fact that prorogation delay has the largest influence in transport networks because of the distance between source and destination, we will only concentrate on measuring jitter because other sources of delay are negligible compared to propagation delay that is not influenced by technologies(Although we can reduce distance my using fiber instead of satellite or direct sight instead of fiber etc. but this is out of the scope of our research).

5 Tools

There are two main sets of tools for measuring delay; hardware tools and software tools

5.1 Hardware tools

There are a variety of hardware tools that do measurements. Most of these hardware are custom made but there are also measurement devices inside network devices as part of OAMs (See [14]). Custom made tools are known for their accuracy but are expensive. There are mainly in two types of custom made hardware tools; active and passive.

Active The active hardware measurement tools send packets on the line and using the sent packets that where sent by the devices measure network characteristics like delay and jitter. Some of these tools can also manipulate specific types of traffic and specific protocols.

Passive The passive hardware measurement tools don't send out packets but listen to the network passively and measure characteristics of the network by time stamping signatures of packets at source and destination.

The problem with these tools is that they might make mistakes because two packets that are alike are not distinguishable by these tools and can cause error in measurement.On the other hand these tools provide characteristics without interfering with actual traffic and operate on an active network

Studying hardware tools in detail is out of the scope of this research but could be done in later studies.

5.2 Software tools

There is a large variety of software tools that have been developed. They are either libraries that other software can use to do measurements or software that can run in kernel space or user space. Extensive research has been done on these tools (See [17]).

Our tool for measurement For our research we are looking for a tool that does OWD. A solution introduced by internet2 is one-way ping (OWAP) (see [8]) based on the One-way Active Measurement Protocol (See [15]).

This solution dose not cover the need to emulate time sensitive applications like VoIP and traffic generation with specific patterns. To achieve this we will use D-ITG (See[1], [4] , [2], [3] and [5]).

D-ITG provides us with the option to generate time sensitive like traffic with the patterns and protocols as used by real applications and it also provides us with options like multiple flows at the same time and multiple senders and receivers.

6 Setups

As discussed before comparing transport technologies by measuring delay is not a suitable solution because of propagation delay. On the other hand measuring network jitter requires that we can distinguish it from the jitter introduced by the two ends of the network that are not part of the network. In our measurements the two ends of our network are our network measurement devices.

In this section we will first build a setup to measure the amount of jitter introduced by the measurements devices. Then we will try to distinguish between the "measurement jitter" and the "network delay" by looking at the "observation delay" in a network. Our desired goal is to be able to measures network jitter in an optical transport network that is transferring time sensitive application packets. To make the setup closer to our desired situation we will generate semi time sensitive traffic, with the patterns and protocols used by real applications, for the measurements.

In our experiments we will emulate VoIP traffic (See table (4) for the traffics specifications) because it is the most commonly used time sensitive application and most tools support emulating it.

As we discussed before, the most suitable solution for measuring delay and jitter is probably using custom built hardware. But because of limitations in resource we can't use hardware solutions for our research. So were are forced to use a software solution. Between the available software solutions, D-ITG best fits our requirements.

Note that in all the setups we use another interface to connect to the machines so that management traffic will not interfere with the experimental traffic.

Because we do not have access to an optical transport network to test our hypothesis, we will try to emulate a transport network.

Transport Protocol	UDP
Codec	G.711 with one sample per packet
Protocol	RTP (Real Time Protocol)
Packet Rate	100 packets/second
Bit Rate	73.6 kbits/second

Table 4: VoIP Traffic Spesifications

6.1 Cross Connect Setup

We know that network devices introduce jitter in a network but network cables don't introduce that. So to measure the jitter introduced by measurement devices ($jitter_i^m$) we can cross connect the measurement ends and measure jitter by them. To do so we cross connected Minsk and Vanilla (See table (5) for specifications) that had D-ITG installed on them (See figure 2).

Device	Specification
Minsk	Pentium Xenon, 3 GHz, 8 cores, 16 GB ram, 1 Gbit Ethernet
Vanilla	Pentium D, 3 GHz, 8 cores, 16 GB ram, 1 Gbit Ethernet
Tomato	Pentium D, 3 GHz, 8 cores, 16 GB ram, 1 Gbit Ethernet
ECP	AMD Athlon 64, 3.4 GHz, 1 cores, 2 GB ram, 1 Gbit Ethernet

Table 5: Device Spesifications

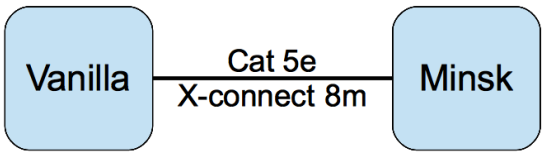

Figure 2: Cross Connect Setup

Then we ran D-ITG for 10 seconds and collected all the $jitter_i^m$s calculated.

We did so by executing the D-ITG server side (ITGRecv) on Minsk and client side (ITGSend) on Vanilla. Then we used D-ITG decoder (ITGDec) to extract jitter data from the log file produced by D-ITG:

```
Minsk side:
./ITGRecv -l crossconectlog.txt -i eth1

Vanilla side:
./ITGSend -a 192.168.10.1 -t 10000 VoIP

Decoding:
./ITGDec crossconectlog.txt -v -j 1 crossconectjitter.txt
```

Finally using an R script we analyzed the jitter data and plotted the figure.

Figure 3 shows the density function [1] of jitter for this experiment.

[1] A density function shows the probability for an specific value in a data set as a function of that value.

As you see in figure 3 most of the jitter are up to 10μs and so we can conclude with an high accuracy that our measurement devices introduce jitters upto that.

Another interesting thing in this figure are the spikes, our speculation is that they are because of the accuracy of our measurements that is probably microseconds but further research is needed to be sure. This speculation is based on the fact that it seems the spikes happen on discrete values.

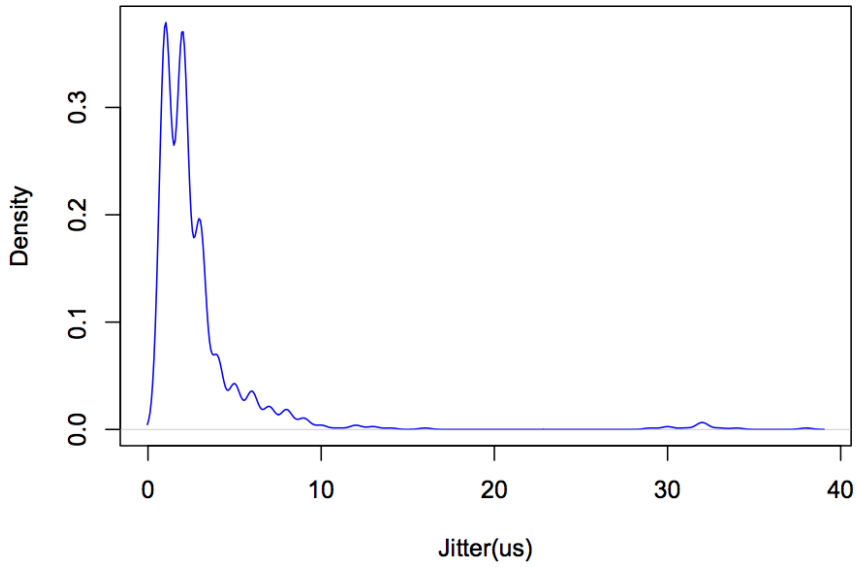

Figure 3: Jitter Density Function for Cross Connect Setup - A

6.2 Virtual environment setup

Now that we know the measurement jitter, we can measure the jitter introduced by the network devices. To do so we were interested to test a optical transport network with a specific technology like OTN or MPLS but we were not able to acquire such a network because of resource limitations. Instead we looked into the feasibility of building such a network in a virtual environment and if it would help us in comparing different technologies.

Based on extensive research done before (See [12]), virtual environments do change results, specially in the orders that we are dealing with (Microsecond measurements), so this would not be a good solution.

Just to see the effect of virtualization on our specific measurements we built two setups, one in the virtual environment and another using physical devices to see its effect.

We built our physical setup by interconnecting Minsk and Vanilla using a Dell Powerconnect 6248 as shown in figure 4.

Figure 4: Physical setup

Then we did the measurements and piloted figure 5 just like the cross connect setup.

Figure 5 shows the jitter density function of the physical setup.

Next we built the virtual environment by running two Xen virtual machines on Minsk (See table (5) for specifications) and interconnecting them using a virtual bridge as shown in figure 6.

And finally we did the measurements and piloted figure 5 just like the setups before. Figure 7 shows the jitter density function of this setup.

Comparing figure 5 and 7 shows a shift toward $10\mu s$ for jitter that indicates that the virtual environment has influenced our measurement of jitter and has added to the amount of jitter. In other words we can see that jitters between $5\mu s$ and $10\mu s$ are more in figure 7 compared to 5. It means that in the case of using a virtual environment, the number of jitters under $2\mu s$ decreases and the number of jitters between $5\mu s$ and $10\mu s$ increases. So we can conclude that measuring jitter using a virtual environment would have side effects and so running doing the experiments virtually would not be a helpful solution.

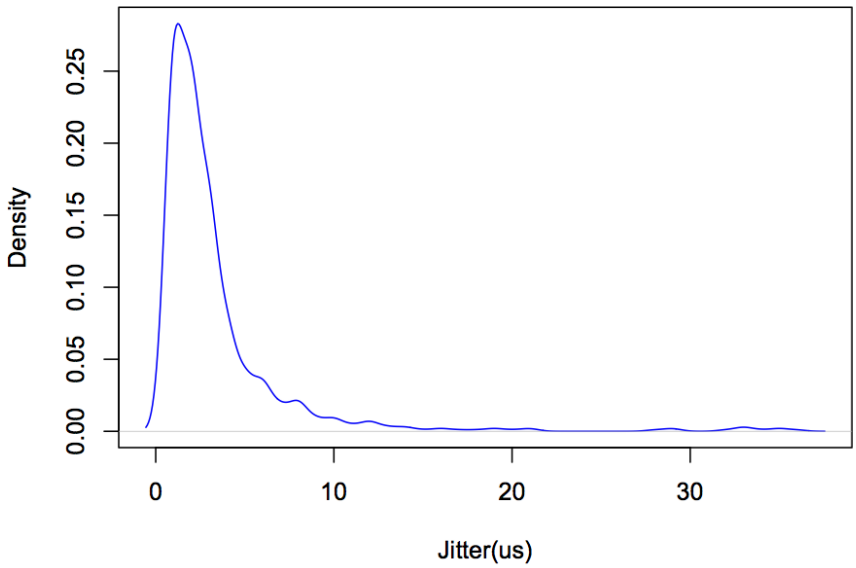

Figure 5: Jitter Density Function for Physical setup

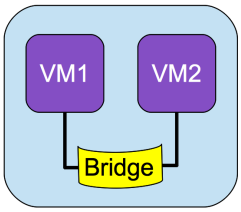

Figure 6: Virtual environment setup

Figure 5 also shows that the switch has shaped the traffic and the spikes that were present in figure 3 are not there anymore.

Further analysis of this figure will require further research because the jiiter introduced by the switch is up to 10μs, just like the measurement jitter, so distinguishing measurement jitter from the switch's jitter is not possible.

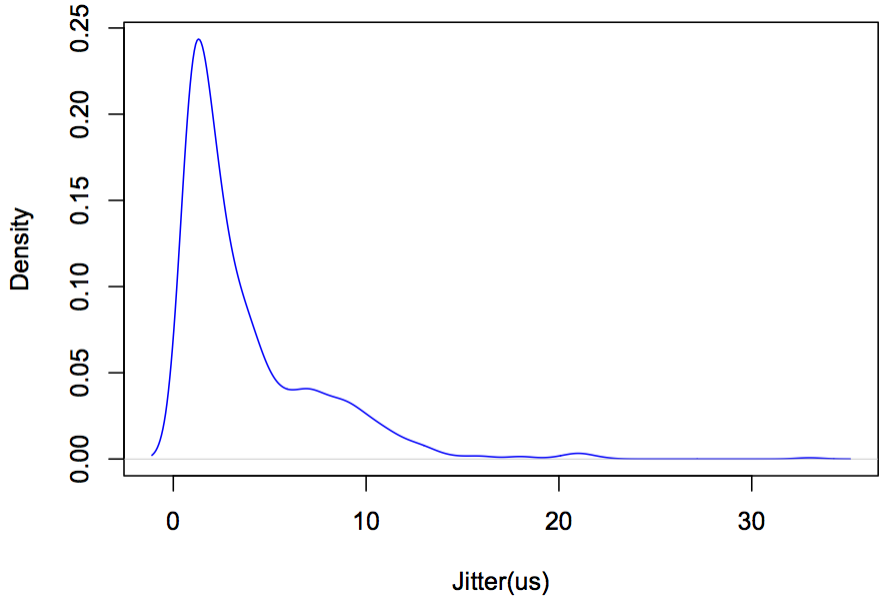

Figure 7: Jitter Density Function for Virtual environment setup

6.3 Emulation Setup

Now that we know that a completely virtualized enviorment is not suitable to test our hypothesis, we can try to prove our hypothesis by looking at the transport network as a black box and emulating it while doing the measurements using physical devices.

To do so we emulate a transport network by adding a device in between our measurement devices and produce delay and jitter in that device. To do so we cross connected Minsk to Tomato and cross connected Tomato to Vanilla and then bridged traffic between the two interfaces of Tomato to build a network as shown in figure 8 (See table 5 for device specifications).

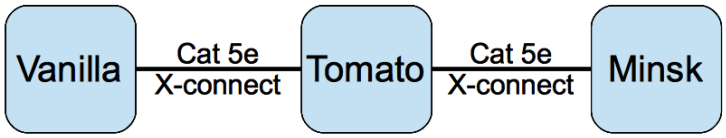

Figure 8: Emulation setup

Then we used "tc" and "netem" in Tomato to emulate 100 ms of delay and 30μs of jitter. We used 100 ms for delay introduced by a 20 km optical fiber and

20 km is about the span of a medium size city.

Figure 9 shows the jitter density function of this setup. In this figure the top of the hill is almost pointing to 30 μs and so we have achieved to estimate the jitter introduced by the emulated network.

The reason we can easily distinguish the 30 μs network jitter in our observation (Figure 9) is that observation jitter is produced by either adding or subtracting measurement jitter from network jitter when talking about specific packet ($jitter_i^o$) [2].

This addition or subtraction only causes a variation around the network jitter in the figure when network jitter is much bigger than measurement jitter and measurement jitter can be considered negligible.

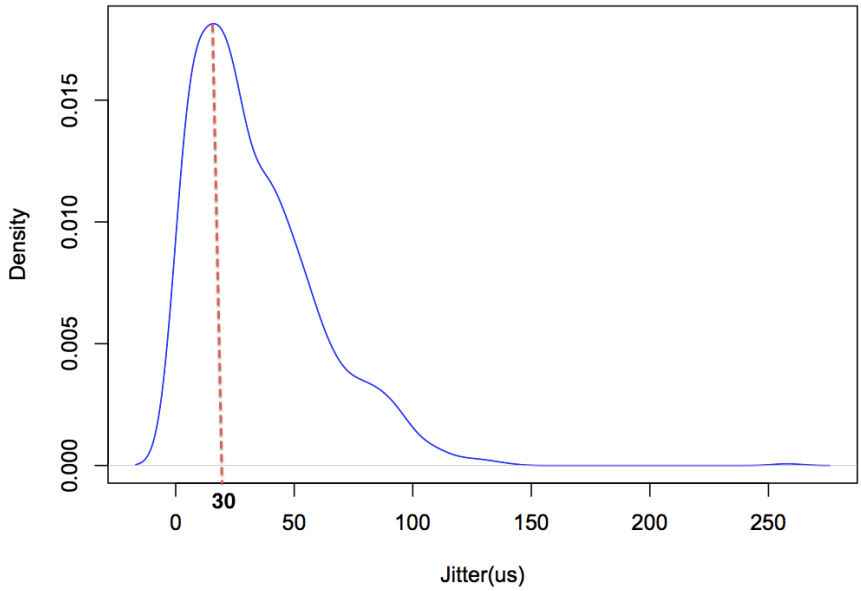

Figure 9: Jitter Density Function for Emulation setup

[2]See section Method

6.4 Wide Area Setup

Now that we have been able to distinguish network jitter by looking at the the density function of the observed jitter, we can try to see how an internet connection that is a mixture of different network devices can contribute to jitter.

To do so we constructed a measurement setup connected Minsk that was located in University of Amsterdam, Amsterdam, Netherlands to ECP that was located in Sharif University of Technology, Tehran, Iran (See table 5 for device specifications) through an ordinary internet connection as shown in figure 10. Then

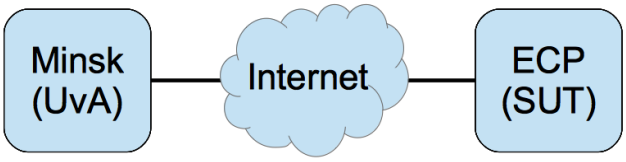

Figure 10: Wide Area setup

we did the measurements and piloted figure 11 just like the setups before, with the exception that this time the D-ITG server was executed on ECP and the client was executed on Minks. Figure 11 shows the jitter density function of this setup.

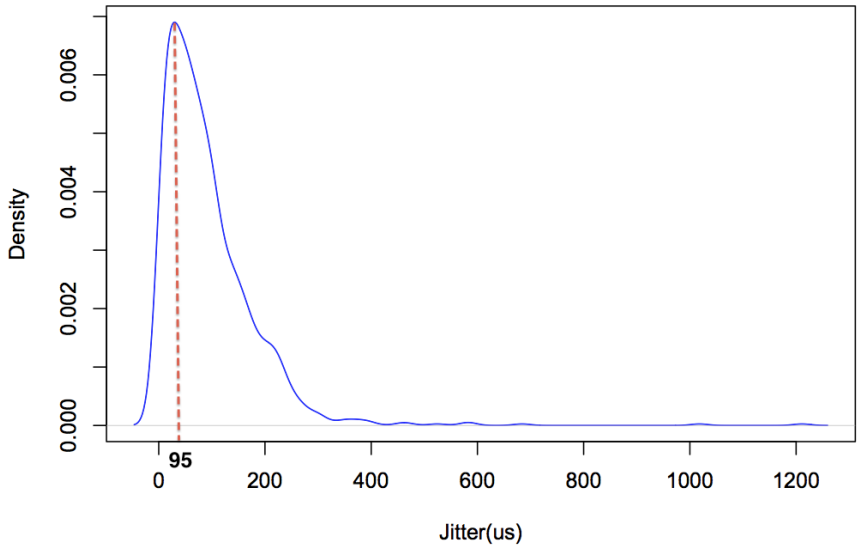

Figure 11: Jitter Density Function for Wide Area setup

We know that in an internet connection we are dealing with multiple transport

switches (mainly MPLS) and multiple routes in the path. Other than these devices, we are also dealing with multiple firewalls that do packet inspection. As we have discussed before one of the sources of delay is processing delay. In a firewall the main cause of processing delay is the packet inspection done by the firewall. Now if processing packets are not all done in a fixed amount of time, this introduces jitter in the network.

When dealing with a firewall that does deep packet inspection, it could be the case that for a set of packet the firewall decides to allow the packets to pass through by just looking into initial characteristics of a packet (like the length of the packet) but for other packets it might require more inspection before letting the packet pass through. This causes packets to have different amounts of delays while passing through the firewall that overall introduces jitter. For example a firewall's policy might be to allow packets with sizes smaller that 70 bytes to pass but it is required that deeper inspection (even until application data) should be done for packets larger than that size. Now if the packets come in random sizes.This can introduce a high variety of jitter.

As you can see in figure 11, the top of the hill of this density function is showing about 95 μs of jitter. This noticeable amount is probably introduced by the firewalls in the path that are doing deep packet inspection. What is also noticeable in the figure, specially in figure 11, is the long tail of the density function, that is about 900 μs wide. That long tail shows the diversity of different amounts of jitter measured. This diversity could have also been introduced by the firewalls but it is important to note that because we do not have control over the complete environment giving a certain conclusion is not possible. For example the diversity of jitter could also be the result of different paths chosen for the delivery of the packets by the routers in between or even sudden changes in the traffic or CPU load of the switches or routers in the path.

6.5 Getting better figures

Although studying density functions for the jitter introduced by each packet is interesting, these density functions have a down side that "one time happenings" influence them. One time happenings influence the figure by causing long tails for the density function. Although one time happenings show the diversity of our results, they prevent us from seeing the details of the common amounts of jitter.

To fix this we can draw the distribution function of jitter for fixed time slices ($jitter^o(\delta t)$) instead of the jitter for exactly each packet.

To do so we run the tests for 2 minutes instead of 2 seconds and then split the results into time slices of 2500 μs , finally we calculate the average for each part and draw the density function for these averages.

We did so by executing D-ITG with the required parameters:

```
Minsk side:
./ITGRecv -l crossconectlog.txt -i eth1

Vanilla side:
./ITGSend -a 192.168.10.1 -t 120000 VoIP
```

and then extracting the jitter averages by executing the D-ITG decoder with the required parameters:

```
Decoding:
./ITGDec crossconectlog.txt -v -j 2500 crossconectjitter.txt
```

Finally using the R script we analyzed the jitter data and plotted the figure.

We did this with the cross connect setup to have a more clear understanding of the reason the spikes were introduced in the figure of this setup setup.

Figure 12 shows the new jitter density function of the cross connect setup.

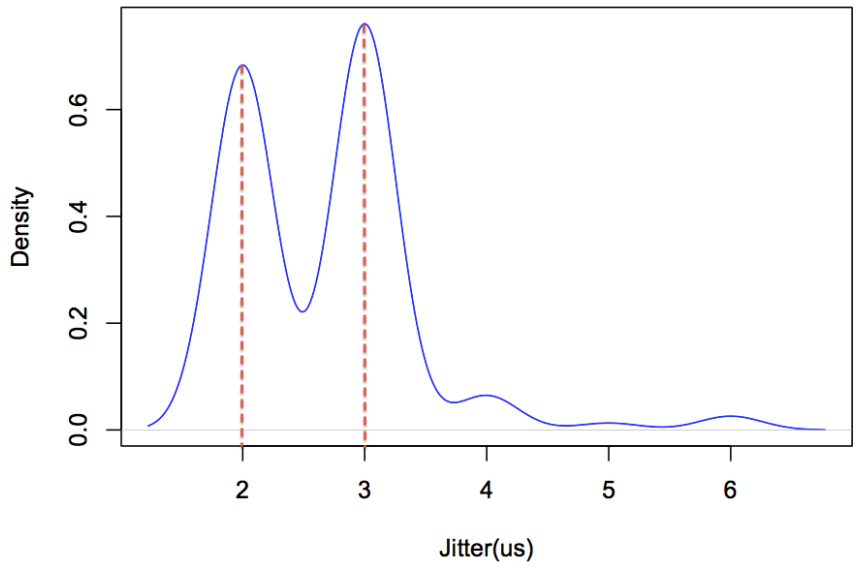

Figure 12: Jitter Density Function for Cross Connect Setup - Fixed Plot

Based on the central limit theorem [3], we should finally end up with a semi-normal density function for jitter that shows the distribution of averages of jitter for fixed time slices. But what we see in figure 12 is two hills. The reason for this is that the size of our data set is not big enough.

[3]The mean of a sufficiently large number of independent random variables each with finite mean and variance will approximately have a normal distribution

In any case if you compare figure 12 with figure 3 you will notice that the spikes in figure 3 have turned into two hills that have peaks in 2 μs and 3 μs. This strengthens our speculation about the accuracy of our measurements to be cretin about our method and to be even more confident about our speculation we executed this method to our wide area setup also.

The parameters for the new wide area setup where set exactly like the parameters for the new cross connect setup.

Figure 13 shows the new jitter density function of the wide area setup.

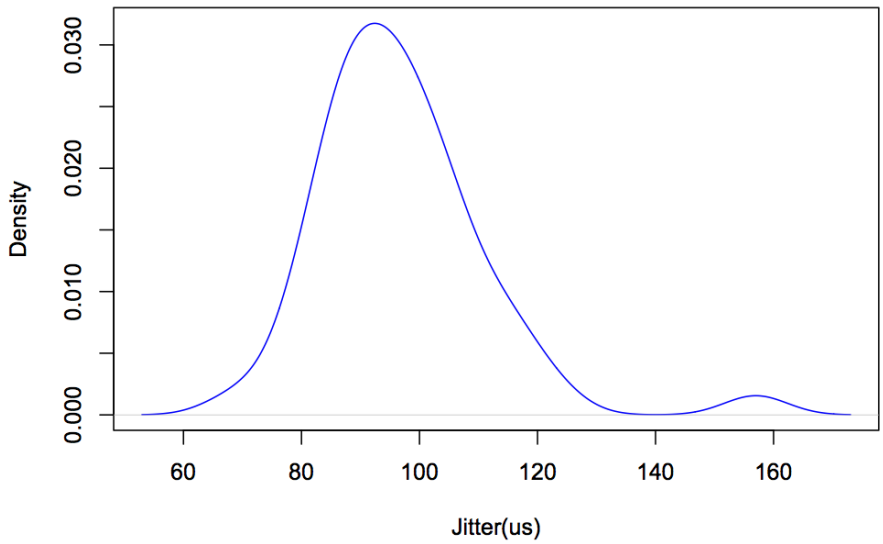

Figure 13: Jitter Density Function for Wide Area Setup - Fixed Plot

As you see figure 13 is quite like a normal distribution except of a small hill at the end of the figure which is probably because our data set is not large enough. Note that the central limit theorem produces an exactly normal distribution when our data set is extremely large. Another reason could be because of the fact that our jitter measurements are not completely independent of each other. In any case, the new figures strengthen our speculation about the measurement accuracy because multiple hills, with almost the same probability of occurrence, were not present in figure 13.

Finally we should note that the parameters used for the setups (2 minute execution and 2500 μs time slices) were chosen by running the test with different parameters and fine tuning the parameters to achieve a reasonable figure.

7 Conclusion

As we discussed, although there are multiple sources of delay most of the delay in a network specially in a transport networks (because of these distance between nodes) is introduced by prorogation delay. We also know that network technologies have minor and maybe even no influence on propagation delay.

On the other hand although other sources of delay like queuing delay and processing delay are negligible compared to propagation delay, they introduce variation of delay (jitter) that propagation delay doesn't introduce.

Jitter has an important role in the perception of delay specially in human interactive application such as voice and video conferencing.

Knowing that measuring jitter will better suite the final goal of comparing different transport technologies for time sensitive applications, we can look into suitable measurement methods and tools for measuring jitter.

Although we think that custom hardware solutions are probably more suitable for these measurement, because of limitation in resources we only studied software solutions.

Between software solutions D-ITG looked most suitable to our requirements.

Finally we conducted a set of experimental setups using D-ITG in order to be able to extract network jittter. Our experimental setups showed that if the jitter introduced by the network in bigger than the jitter introduced by our measurement device we are able to distinguish the network jitter by having the observation jitter.

We then conducted an experiment on a wide area network and looked into the effects of firewalls on the introduction of jitter in this network . Finally we used the central limit theorem to solve the problem of long tails in our distribution functions that are a result of one time happenings. Using our new distribution functions we were able to get more details about jitter in the network. From the results of the new distributed functions we got almost certain that D-ITG's measurement accuracy is probably $1\mu s$ and that the central limit theorem is working correctly with the parameters we have chosen to do our experiments.

In conclusion as we noted the most suitable parameter for comparing transport technologies for time sensitive applications is jitter and the experiments should be conducted using a measurement device that introduced the minimum amount of jitter. It is also important to note that the more complex our experimental setup is the more difficult it will be to detect the actual cause of jitter. So to decide between technologies or devices we should keep our setup as simple as possible and prevent complexity that might introduce dependence between jitters introduced for two separate packets.

8 Future Work

In this research we tried to emulate an optical transport network. However it would be interesting to do the experiments with a real optical transport technology and compare the results with our research.

Another part that could be added to this research is adding queuing theory that will help us understand queuing delay better.

Other than that research can also be done on comparing different hardware solutions and the implementation of OAM in different network devices.

Although a lot of research has been done in this field, we believe that there is still a lot of uncovered areas that could be conquered.

References

[1] Leopoldo Angrisani, Domenico Capriglione, Luigi Ferrigno, and Gianfranco Miele. Packet jitter measurement in communication networks: A sensitivity analysis. *2011 IEEE International Workshop on Measurements and Networking Proceedings (M&N)*, pages 146–151, October 2011.

[2] Stefano Avallone, Alessio Botta, Alberto Dainotti, Walter De Donato, and Antonio Pescapé. D-ITG V. 2.7. 0-Beta2 Manual. Technical report, 2009.

[3] Stefano Avallone, Donato Emma, and Antonio Pescapé. A practical demonstration of network traffic generation. *Proc. of the 8th IMSA*, 2004.

[4] Stefano Avallone, Donato Emma, Antonio Pescapé, and Giorgio Ventre. Performance evaluation of an open distributed platform for realistic traffic generation. *Performance Evaluation*, 60(1-4):359–392, May 2005.

[5] Alessio Botta, Alberto Dainotti, and Antonio Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, March 2012.

[6] C. Demichelis and P. Chimento. RFC 3393 : IP Packet Delay Variation Metric for IP Performance Metrics (IPPM), 2002.

[7] Joel Hasbrouck and Gideon Saar. Low-Latency Trading. *Manuscript, Cornell University*, 10012(September), 2011.

[8] Internet2. One-Way Ping (OWAMP): An Internet2.

[9] Gunnar Karlsson. Asynchronous Transfer of Video. *IEEE Communications Magazine*, pages 118–126, 1996.

[10] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Pearson Education Inc., 2010.

[11] D Mills, U Delaware, J Martin, J Burbank, and W Kasch. RFC5905 : Network Time Protocol Version 4: Protocol and Algorithms Specification, 2010.

[12] Rafika.I Mutia, N Sadeque, J.A Anderson, and A Johnson. Time-stamping accuracy in virtualized environments. *(ICACT), 2011 13th*, pages 475–480, 2011.

[13] John Nagle. RFC 970 : On Packet Switches With Infinite Storage, 1985.

[14] Jeong-dong Ryoo, Jongtae Song, Jaewoo Park, and Bheom-Soon Joo. OAM and Its Performance Monitoring Mechanisms for Carrier Ethernet Transport Networks. *Communications Magazine*, pages 97–103, 2008.

[15] S Shalunov, B Teitelbaum, A Karp, J Boote, and M Zekauskas. RFC4656 : A One-way Active Measurement Protocol (OWAMP), 2010.

[16] Ralf Steinmetz. Human perception of jitter and media synchronization. *Areas in Communications, IEEE Journal on*, 14(1), 1996.

[17] Daniel Turull Torrents. Open source traffic analyzer, 2010.

A R Script

```
setwd("data/")
getdelaydata<-function(server,nonzero=TRUE){
dat = read.table(paste("delays/",server,".txt",sep=""))
if(nonzero==TRUE){
dat=dat[dat$V2!=0,]
}
dat[,2]
}
getjitterdata<-function(server,nonzero=TRUE){
dat = read.table(paste("jitters/",server,".txt",sep=""))
if(nonzero==TRUE){
dat=dat[dat$V2!=0,]
}
dat[,2]
}

drawjitterhist<-function(server,bre=100,nonzero=TRUE){
dat = getjitterdata(server,nonzero)
dat = (dat)*(10^6)
hist1<-hist(dat,right=TRUE,main=paste("Histogram of Jitter for",server )
,col="blue",xlab="Jitter(us)",breaks=bre,xaxt='n')
axis(1, hist1$breaks)
}
jitterhistdata<-function(server,bre=100,nonzero=TRUE){
dat = getjitterdata(server,nonzero)
dat = (dat)*(10^6)
hist1<-hist(dat,right=TRUE,main=paste("Histogram of Jitter for",server )
,col="blue",xlab="Jitter(us)",breaks=bre,xaxt='n')
hist1
}


drawdelayhist<-function(server,bre=100,nonzero=TRUE,fix=0){
dat = getdelaydata(server,nonzero)
dat = (dat*(10^3))+fix
hist1<-hist(dat,right=TRUE,main=paste("Histogram of Delays for",server )
,col="blue",xlab="Delays(ms)",breaks=bre,xaxt='n')
axis(1, hist1$breaks)
}
```

```
delayhistdata<-function(server,bre=100,nonzero=TRUE,fix=0){
dat = getdelaydata(server,nonzero)
dat = (dat*(10^3))+fix
hist1<-hist(dat,right=TRUE,main=paste("Histogram of Delays for",server )
,col="blue",xlab="Delays(ms)",breaks=bre,xaxt='n')
hist1
}
drawjitterdensity<-function(server,nonzero=TRUE,kernel="gaussian"){
    dat = getjitterdata(server,nonzero)
    dat = (dat)*(10^6)
    plot(density(dat,kernel=kernel)
    ,main=paste("Density function of Jitter for",server )
    ,col="blue",xlab="Jitter(us)",col.lab="black",col.main="black"
    ,col.sub="black"
    ,col.axis="black",fg="black")
}
drawdelaydensity<-function(server,nonzero=TRUE,fix=0
,base=FALSE,kernel="gaussian"){
    dat = getdelaydata(server,nonzero)
    dat = ((dat)*(10^3))+fix
    plot(density(dat,kernel=kernel)
    ,main=paste("Density function of Delay for",server )
    ,col="blue",xlab="Delay(ms)")
}
drawjitterall<-function(server,bre=100,nonzero=TRUE,kernel="gaussian"){
dat = getjitterdata(server,nonzero)
dat = (dat*(10^6))
hist(dat,right=TRUE,prob=TRUE
,main=paste("Histogram of Jitters for",server )
,col="blue",xlab="Jitters(us)",breaks=bre)
lines(density(dat,kernel=kernel))
}
drawdelayall<-function(server,bre=100,nonzero=TRUE,fix=0,kernel="gaussian"){
dat = getdelaydata(server,nonzero)
dat = (dat*(10^3))+fix
hist(dat,right=TRUE,prob=TRUE
,main=paste("Histogram of Delay for",server )
,col="blue",xlab="Delays(ms)",breaks=bre)
lines(density(dat,kernel=kernel))
}
```

```
jittersafty<-function(server,nonzero=TRUE){
dat=getjitterdata(server,nonzero)
stadev=sd(dat)
avrg=mean(dat)
print(paste("sd:",stadev))
print(paste("mean:",avrg))
print(paste("68%:",avrg+stadev))
print(paste("97.5%:",avrg+(2*stadev)))
print(paste("99.7%:",avrg+(3*stadev)))
}
delaysafty<-function(server,nonzero=TRUE){
dat=getdelaydata(server,nonzero)
stadev=sd(dat)
avrg=mean(dat)
print(paste("sd:",stadev))
print(paste("mean:",avrg))
print(paste("68%:",avrg+stadev))
print(paste("97.5%:",avrg+(2*stadev)))
print(paste("99.7%:",avrg+(3*stadev)))
}
```