



ARITARI  
NEW WORLD NETWORKS



# ARITARI VOICE OPTIMISATION

WHY MORE BANDWIDTH IS NOT THE ANSWER



# TABLE OF CONTENTS

<b>INTRODUCTION .....</b>	<b>3</b>
London Calling.....	6
Electronic Exchanges .....	6
Measurement of How Bad it Really Is (MOS).....	10
<b>(SOME OF) THE PROBLEMS WITH VOIP.....</b>	<b>11</b>
You've got your Head in the Sand!.....	11
Link Failure.....	11
Delay.....	12
Jitter.....	12
Packet Loss.....	12
Inefficiencies.....	13
<b>HOW ARITARI VIBE HELPS .....</b>	<b>14</b>
<b>TEST RESULTS.....</b>	<b>15</b>
Unaccelerated Network.....	15
Aritari's Solution.....	23
<b>SUMMARY.....</b>	<b>31</b>

## INTRODUCTION

Whilst network connectivity has vastly improved in terms of speed and reliability in recent times, the ability to utilise the available bandwidth efficiently has not really kept pace with these changes. Aritari's ViBE SD-WAN/VPN was developed in order to redress this balance, and to make the modern network more efficient and controllable whilst giving higher visibility and availability when problems occur.

This document will focus on how an Aritari SD-WAN or VPN enhances voice communication over wide area networks, and allows those networks to be used for simultaneous voice and data without compromise, and without sacrificing audio quality. It also includes some hopefully interesting background information to explain *why* we can achieve such impressive results. The last section shows comparisons of with and without ViBE in the mix, which really speak for themselves.

Initially developed with Voice over IP (VoIP) in mind, over the years ViBE has extended its reach to all traffic sent over the WAN whilst retaining the initial philosophy of allowing existing infrastructure to fulfil its maximum potential.

The advantages of using ViBE are many, but include:

- Faster throughput when using TCP connections, which are the majority of connections in use on the Internet today.
- Much less impact on network performance when transient or more permanent issues occur.
- Instant notification of degraded performance and the near real-time visibility of network parameters such as latency, loss, bandwidth and predicted MOS score for G.711 VoIP.
- Full central control and configuration ( "orchestration" ) of network and device operation, so that customer CPE devices can be provisioned and configured centrally. This includes all aspects of the CPE, such as firewall and DHCP server, for example. There is also the possibility to automatically generate device configurations based on customer requirements.
- Much more efficient use of bandwidth, especially VoIP and transactional data which typically uses small packets of information, but also with file transfers and other applications.
- Near-instantaneous switch to backup networks with no loss of ongoing connections in the case of a degraded or failed main network. The main and backup networks can be over any combination of any medium, for example fibre, satellite, cellular data or DSL. Default configuration results in around a second of interruption when a major link fails, though this can be configured to be much less if required (but in any case, IP addressing etc. remains consistent, so no connections are lost.)
- Bonding of multiple connections of varying bandwidths and technologies, with the ability for any single data stream to use all available throughput on all links simultaneously. Individual links are monitored and can automatically be taken out of service if certain performance parameters aren't met, and then restored once the quality improves above a certain threshold.
- Packet loss mitigation using multiple links simultaneously (RAIN mode) or a single link with redundant data. This can be combined with bonding or only used for certain types of traffic, such as voice.
- Optional encryption of all traffic. Note that this is disabled by default, on the basis that pretty much everything sent on the WAN is encrypted anyway these days and adding an additional layer of encryption would be pointless.
- Optional traffic reduction over multiple routes. This feature allows multiple endpoint addresses to be used for a single connection, thus splitting all traffic over multiple routes and making it impossible to monitor that traffic for any single third party. Each endpoint address is individually monitored to ensure that connectivity is available.
- Optional ViBE transport protocols. The default is to use UDP, but occasionally and in specific circumstances service providers and network backbone providers can introduce restrictions to the flow of UDP, so it's possible to configure ViBE to use TCP or even ICMP as its transport protocol.
- Complete support for operation through NAT - either the CPE device or server can be behind a NAT device.
- The ability to be deployed on physical hardware, or virtual machines/cloud infrastructure such as AWS or Azure, and as a managed service or white-label product, as well as individual deployment within customer infrastructure.
- Our own CPE devices have full router capability including firewalling, policy routing, and so on. We also support specific 3rd party hardware from manufacturers such as Mikrotik and iQsim.
- Optional full layer 2 support, including the ability to transport VLAN and double-tagged VLAN traffic through the tunnel. ( V7, in testing phase. )
- Optional data deduplication and compression support (V7, in testing phase.) This requires that data through the tunnel is NOT encrypted, so would only be of use in cases where both ViBE endpoints are on user-controlled networks, so that all encryption of user data can be disabled and ViBE tunnel encryption enabled in order to maintain privacy.

## BRIEF HISTORY OF VOICE (NETWORKS)

From the moment humans discovered how to utilise electricity for more than just power, it was inevitable that it would be used as a means to communicate with others across the globe. The invention of the telephone itself is shrouded in confusion and many people have claimed responsibility at various times, but there can be little doubt that legally speaking, Alexander Graham Bell's patents were pivotal in its early development, since the US courts decreed that they were valid, and the ideas within were used as the basis for early devices. In reality it's likely that many had similar ideas, since it was a fairly obvious extension of the electrical telegraphs which had been around since before the mid 19th century. These telegraphs could already signal using electricity over a distance, but were text based and used on-off type signals to denote characters.



## London Calling

Early voice networks were simply pairs of wires, along which an electrical signal proportional to the speaker's voice was passed. A call was made by creating a physical circuit between the maker and the receiver of the call, so that the signal put in at one end via a microphone and amplifier was passed over the wires, and then amplified again at the receiver and sent to a speaker.

Clearly it wouldn't be very useful or practical for every user to be connected to every other by their own pair of wires, so there were manual exchanges with trunk routes between them. Every user's own pair of wires fed into a switchboard at one of these exchanges, where an operator would use patch cables to connect the caller to where they wanted to go (either to another user connected to the same exchange, or to a trunk route to another, where another operator would forward the call on. ) It could take many minutes to connect a long-distance call in this fashion, because every single exchange along the path needed an operator to work out where to forward the call next, and to make the appropriate connection.

The next real major development was by Almon Brown Strowger, who gave his name to the Strowger Switch. This was an electromagnetic device controlled by a pulsed voltage, which could be created by a telephone device at the caller's end of a line. Those of a certain age may remember the rotary dial telephones, whose dialing device consisted of a disc with holes representing each digit, 0 - 9. You would insert your finger in a hole, and rotate clockwise until the disc hit a stop, and then release - the result being a set of disconnections of the telephone line from the device creating "off" pulses, the number of which matched the number of the digit dialed (with 0 being 10.) The first digit dialed would cause the Strowger Switch on the end of the line to step up that number of levels in its mechanism, then the next would cause it to rotate, thus connecting the input (the customer's line) to an output, which could then be connected to another switch, a trunk route to another exchange, or another user on the same exchange. Since the number of these switches in any given exchange was limited, it was quite common to find the trunk route already in use, and it was also common practice for multiple users to have to share the same switch connection.

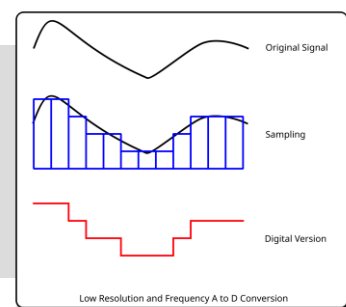
## Electronic Exchanges

Electronic exchanges (or at least the concept of them) began to appear in the early to mid twentieth century, though in the early days they were essentially just replacing the expensive and error prone Strowger switches with electronic versions - the signal carried from speaker to listener was still very much analogue in nature. True digital exchanges had to wait until a number of technologies were ready, the first of those being digital processors themselves, and also a way to convert the analogue (proportional) voice signal into one that a digital processor can handle.

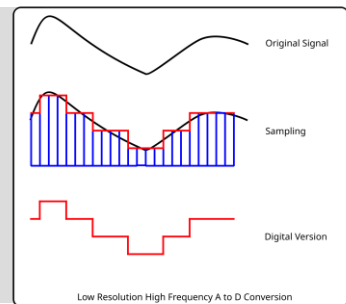
## Analogue to Digital Conversion

This process is called (logically enough) analogue to digital conversion, and is carried out by an analogue to digital converter or ADC. In essence, it involves looking at the signal at a point in time and reading its voltage level, which can then be represented as a binary number. This is repeated many times per second, resulting in a sequence of binary numbers which represents the original signal, though how closely that is true depends on how many times per second you take a sample, and how many different numbers you use between the lowest and the highest voltage you may find (the resolution.) In the simplest case, each number is the same size in computing terms - so if you had 1000 different levels, the "0" would actually be 000, with maximum being 999. Clearly, the more times you take a sample, and the bigger each value is, the greater the number of digits used to represent the original signal, and the closer to that original signal you get.

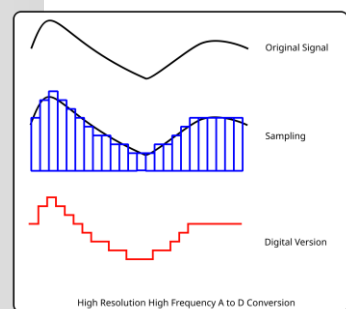
This diagram shows what happens when you sample at a low frequency and low resolution - each resulting sample is small (because there are a small number of levels) and the rate is low, so the amount of data needed to store or transmit the digital signal will be small, but as you can see from the red line, the result doesn't look very much like the original signal.



If we increase the number of samples (in this case we sample at twice the rate we used before) then each sample remains small, but there are twice as many of them, so the amount of data needed for the signal is double what it was before. This has resulted in a digital version of the signal which is a little closer to the original, but it's still pretty poor - we can track changes more quickly (i.e. capture higher frequency signals) but not get any closer to the actual level at each sample point.



Finally, we can also double the number of levels we use, i.e. double the resolution. Since computers work in binary, doubling the resolution only adds one bit to each sample ( a 4 bit number can hold 16 values, 0000, 0001, 0010, ... 1111 but a 5 bit number can hold 32. ) This means that doubling the resolution requires less of an increase in the data rate, and gets us closer again to the original signal. Increasing the resolution allows each sample to be closer to the original, but we can't store signals with a higher frequency (rate of change) without further increasing the number of samples we take.



The opposite process, digital to analogue conversion, is also, of course, required, since not many humans can understand a digital signal, preferring instead to get the good old analogue signal from a speaker. The quality of the voice you hear will depend on the sample rate and resolution of the original signal (you can't get back what you've lost) and also the quality of the filtering used.

## Encoding Voice (the CODEC)

In the simplest case, each level of an analogue signal could be represented by a number which is a linear representation of that original signal, which is what we did with the sampling examples above. For example, if you're sampling a signal which could range from 0 to 4 Volts, you could use 256 levels (computers normally use powers of two) with 128 representing 2V, 192 is 3V, and so on. However, that would result in a digital stream which isn't *that* close to the original - 0.1V for example would be represented by the number 6, but when you convert 6 back to analogue using the same method you find that it *actually* comes out as 0.09375V, again as you can see above. This introduces digital noise, which when passed through the DAC creates analog noise - essentially a random seeming deviation from the original.

On the other hand though, you can't simply use an infinite number of levels in the sampling process - for one thing infinity tends to be quite a tricky amount of anything to handle. Therefore there has to be a compromise between the quality and the amount of data needing to be sampled (and hence transmitted.) In order to make this compromise the best it can be, the early digital pioneers realised that there are certain features of the human voice and hearing that they could take advantage of:

- Humans are much more sensitive to small changes in level when the starting level is already small... that is to say, detail in loud sounds needs to be much more pronounced to be perceived.
- The vast majority of the audio in a human voice is between 300Hz and 3.4kHz.

It can be shown that the frequency response of an analogue to digital conversion process is roughly half the sampling rate, so as a consequence 8kHz (8000 samples per second) was selected as an adequate sampling rate for speech (theoretically allowing up to 4kHz to be represented.)

In order to reduce the resolution required for a "good" signal, a linear representation isn't used. Instead, the number of levels representing the lower signal voltages is greater than the number at higher ones, so taking advantage of the differing sensitivity of hearing. In effect, what actually happens is that a voice signal is sampled with a large range of values, which is then encoded into the smaller range using an algorithm which removes possible levels from the higher voltage ranges. The opposite then happens at the receiving end (i.e. the signal is decoded before being converted back to analogue.) The algorithm used to encode and decode the signal in this way is known as the CODEC (enCOder/DECoder.)

There were two main variants of the CODEC used in these earlier digital systems, known as A-law (mainly Europe) and  $\mu$ -law (mainly the US.) Both work on the same principle, but have a slightly different algorithm - A-law tends to give more levels to the lower voltages than  $\mu$ -law. The resulting signal in both of these is an 8 bit (0 - 255) value, and at 8000 samples per second this results in 8000 bytes per second or 64 kilobits per second.

Both of these variants are incorporated into a standard called **G.711**.

G.711 was still widely used in digital telephone networks until recently, though with the advent of Internet based telephony more efficiency and latterly higher quality was required.

## Transmission and Switching

Another key component of a truly digital telephony system is the ability to send the binary stream representing a voice between exchanges. This is where Asynchronous Transfer Mode or ATM comes in - it's the protocol used to carry such signals (and, as it happens, generic data, a fact which will become important later.) ATM was developed very much with voice, and hence the original "circuit switched" nature of the telephone exchange, in mind.

As a simple overview, ATM essentially replaces the multiple copper pair trunk routes of those original manual switchboards. Before two ends of a connection can communicate, a "virtual circuit" needs to be created between the sender and receiver, mimicking the actions of the plugboard operators at the exchanges. Once this is done, a point to point connection is made and hence anything sent at one end is received at the other, wherever those two places happen to be, and the path taken by each subsequent packet of data is the same.

Each link along the route has a fixed transmission rate and hence can carry a fixed number of G.711 calls - each call and the information about its virtual circuit is "multiplexed" with other calls on the same link.

To illustrate this, let's take an example of a technology that many will be familiar with, ISDN (International Subscriber Digital Network.) A "basic rate" ISDN connection actually allows two simultaneous calls to be made - this is because it has two data channels at 64 kbit/s each (remember that rate?) plus a signalling channel. The actual voice data will consist of "cells," which are chunks of data of 53 bytes, with 48 bytes available for voice. In a basic rate circuit, the cells from the two channels will be interleaved on the wire (along with a 16kbit/s channel for signalling.) The use of such small cells means that any delays due to congestion are small, but in any case the fact that a virtual circuit is created to the destination means that the bandwidth is effectively reserved end to end - once a connection is established it is guaranteed (barring any failures.)

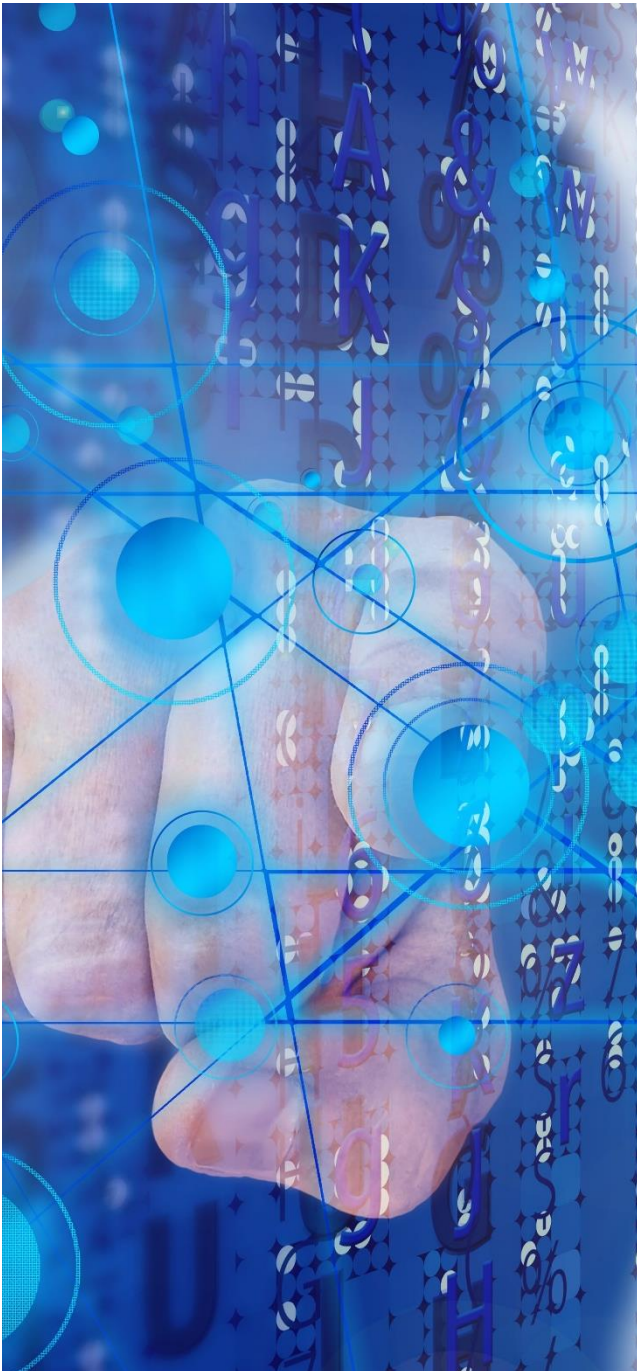
## The Advent of the Internet and Voice over IP

The Internet as we know it today has its origins in the early 1970's, but the key characteristic as far as we're concerned is that it was conceived as a network for carrying data of any description, to anywhere, and so is a packet switched network. Unlike a voice network, on which an end to end circuit has to be established in order for communication to take place, each individual chunk of data (packet) sent on the Internet needs to be able to find its way, by whatever route, to its destination. In terms of information getting from the sender to the receiver, this leads to several important concepts:

- Each packet needs to contain enough information for it to get to where it needs to go.
- As far as the network is concerned (as opposed to the devices on it,) any packet of information is completely independent of any other - that is to say that left to its own devices, two packets between the same source and destination may actually travel along different paths to reach their target.
- The packets need to be large (relatively speaking when compared to ATM cells) because of the need for efficiency. Small packets still contain all of the routing information, so for the application the ratio of "useful" (payload) to "useless" (signalling) data is poor.

This is all quite different from the principles of a voice network, and the very nature of a packet switched rather than circuit switched network when applied to real time voice creates all sorts of problems, even though in principle digital voice samples are encoded into a simple data stream as a first step in both cases.

The underlying protocol used for the Internet is (logically enough) Internet Protocol (IP) and voice carried using it is Voice over IP (VoIP.)



## Measurement of How Bad it Really Is (MOS)

Given how unsuited to voice communication the Internet is, a method was needed to measure just how bad the voice quality at the receiving end actually is. To this end, in the early 1990's it was decided that the best way to find this out was to ask people, so that's what happened. This was the genesis of the "Mean Opinion Score" or MOS, and it was just that - a simple questionnaire was given to a group of people who took part in a conversation. The questions were about how good they thought the call was in terms of intelligibility and how "nice" the call sounded, each answer being a number from 1 to 5 (where 5 is excellent and 1 is bad.) These answers were then collated to give an aggregate score, also in the range of 1-5.

Before even sending a call over the network, the CODEC used has an impact on the call quality. For example, the "best" encoding method that was available for many years was G.711, which obtained a MOS of 4.4 - for comparison G.729 manages only 4. These are the starting point for how bad the call is... they only go down from there.

You will often see measurement tools that claim to show you what the MOS of a call is (based on CODEC used, packet loss, jitter etc) but in reality, as the name suggests, the score is the opinion of people, and (for now at least) an algorithm doesn't have an opinion. However, these scores at least give some idea of call quality, and are much easier to obtain than getting a bunch of people to try it.

## (SOME OF) THE PROBLEMS WITH VOIP

As we've already seen, a packet switched network such as the Internet is quite different from a circuit switched one (the original digital telephone network.) Anyone who's had any sort of dealings with VoIP will have heard all of these terms before, because they're always present to a lesser or greater degree, even if the network and telephony devices connecting to it try to do their best to mask them.

### You've got your Head in the Sand!

First and foremost the issues below happen. They happen ALL... THE... TIME. It may be that 99% of the time, the quality of your calls is ok, and that odd glitch here and there is "just one of those things." It may be that your customers are having trouble hearing you, but hey, that's "just one of those things." When your network is down for hours and you've lost thousands of dollars worth of business, that's "just one of..." - no, wait a minute...

The fact is that people have come to expect poor quality calls, so to a certain extent issues are tolerated... apart from the network failure, but that will never happen, will it? Will it?

### Link Failure

One of the most important issues with voice over the Internet (or indeed voice calls generally) is the question of what happens when a link fails. Often individuals and even businesses rely on a single main connection to the Internet, so if that fails all bets are off. However, even those that implement resiliency often give little thought to how their voice network will cope. At the very least, it's likely that the IP address seen as the source address for calls will change, causing at best all current calls in progress to drop, but maybe even registration issues with servers and other equally serious problems. The best case is often that users are incommunicado for minutes, but it can be much worse.



## Delay

Strictly speaking, this isn't a problem specific to the Internet, but one affecting all long distance networks - electrical signals essentially travel at the speed of light, which is finite, so any real-time conversation between someone in the UK and Australia, for example, will be subject to a delay (in that case of around 120mS.) This gets much worse if the communication is via geostationary satellite, in which case delays of 300mS or more can often be experienced. In addition, it will take a period of time for the sending device to encode speech into a data stream, depending on exactly which CODEC is used and the type of hardware involved.

The issue specific to the Internet is that there are almost always additional delays experienced by an IP packet on its journey, whereas on a circuit switched path these delays are close to the minimum possible.

It's generally accepted that the maximum one-way delay on a voice call is around 150mS - beyond this figure it becomes noticeable and the conversation gets disjointed (people talking over each other etc.)

Additional delays in packet switched networks happen for many reasons, but the fixed additional latency is generally because of the time it takes for all routers in the path to receive, process, and then transmit a packet, plus the time it takes to encode and decode the voice packets - the encoding delay for G.729 for example is at least 15mS.

## Jitter

Jitter simply refers to the fact that the delay experienced by each packet sent over a packet switched network will be different, and it's this difference that constitutes the jitter value. Devices that process VoIP have to compensate for this jitter by storing enough packets so that there is always enough data to play back at a constant rate (in what is known as a jitter buffer,) so the net effect of jitter is in fact a further delay on the call.

One main source of jitter is at any point where one network link (or the sum of several links) feeds into another link which has a lower capacity. At these junctions, routers need to keep hold of packets until there is space available to send them, meaning that the additional delay will depend on how many packets are already waiting to be sent, and how the router prioritises those packets (if at all.) Note that this doesn't mean that the slower link is over-utilised, just that in that instant a number of packets are waiting to be sent, a situation which can occur for very brief periods on any link. This additional delay due to jitter can become problematic, especially if the fixed delay mentioned above is already close to the threshold at which it's noticeable.

## Packet Loss

If any packets go missing between the sender and receiver, then obviously this results in some of the voice data being lost. Whilst early CODECs can be affected by even a small percentage loss, later ones tend to have a packet loss concealment mechanism which can tolerate a small percentage loss by approximating what's missing, but even so the ideal case is to have no loss at all.

The problem is that the packet queues within routers that are used to buffer packets ready to be sent on a network tend to be kept quite small so as to minimise any delay caused by them filling up. This means that it's quite possible (likely even) that some will be simply thrown away, because the buffer is already full when they arrive.

## Inefficiencies

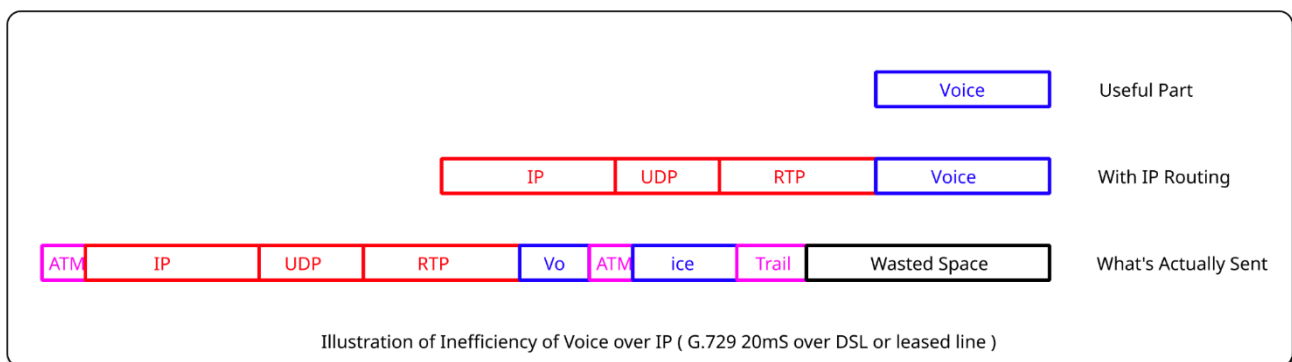
As has already been mentioned, each data packet on the Internet needs to contain all of the information needed for it to get to its destination. This additional routing information will include network, protocol and transport layer overheads, which can be quite large.

<b>Physical Layer</b>	These overheads are to do with how signals are sent on the network itself, which for Ethernet is around 20 bytes.
<b>Data Link Layer</b>	This information is used to get a packet between two hosts on a single network segment, such as an Ethernet network. In the case of Ethernet, this is a minimum of 18 bytes. However, many wide area links are still carried over an ATM network, and as previously described these use fixed 53 byte cells with 5 bytes of routing information in each. The IP version of ATM then splits a packet up into 48 byte chunks to fit in however many cells are needed, plus an additional 8 bytes added to the end of the packet (so effectively the last cell can only hold 40 bytes of data, rather than 48.)
<b>Network Transport layers</b>	These are for the Internet protocol itself, as well as the transport protocol which for VoIP is usually UDP. This will add an additional 28 bytes.

In a typical file transfer, each packet will contain 1400+ bytes of data, so these overheads don't cause too much of a headache. However, when you consider voice it's a completely different story, because VoIP packets must be kept relatively small so that issues of delay, jitter and loss are minimised. Until recently, the most popular CODEC used for voice calls was G.729, chosen because of its efficiency and decent quality when compared to G.711. The resulting datastream after encoding is only 8kbit/s, which is split into 10, 20 or 30mS (usually) chunks. However, let's consider what happens when a G.729 call ( at, say, 20mS packet size ) is sent over a DSL link.

- Each 20mS chunk has 20 bytes.
- RTP adds 12 bytes
- UDP adds another 8 bytes,
- IP adds 20 bytes.
- ATM adds 8 bytes initially, giving 56 bytes in total. It then splits that into two parts, the first of which is 48 bytes to fill one cell, and the second is *also* 48 bytes, because a cell *has* to contain 48 bytes. Each cell *also* has 5 additional bytes.

After all of this, the original 20 byte chunk of data gets sent across the network as 106 bytes (two ATM cells) - so since you have 50 of those every second that amounts to 42.4kbit/s, 34.4 of which is overhead! Here's a visual representation, so you can see just how ridiculous this really is!



## HOW ARITARI ViBE HELPS

When a ViBE tunnel is overlaid on an existing network, all traffic passing through that tunnel is under its control. Since it was designed from the ground up with VoIP in mind, it removes all of the issues listed in the previous section and also provides additional tools if the underlying network itself is at fault. Whilst the details of how it works are proprietary, essentially it:

Allows redundant links to easily be configured to provide backup routes for calls and other Internet traffic, with the default “out of the box” settings giving a fraction over a second of audio loss with no dropped calls or requirement for telephony devices to re-register. These redundant links can use any technology, and in fact given the other features of the Aritari ViBE SD-WAN even 4G can be used to give a usable backup for quite sizable call centre operations, and a simple DSL link could service pretty much any sized business. For those with a need for absolute zero downtime, redundant connections can be configured as mirrors, meaning that a service loss on a single link has no effect whatsoever. Note that in the case where more than a single link is configured as an active-active ( bonded ) connection, there will still be a one second interruption if one of them fails, though there would be packet loss during this time rather than complete silence ( since the other link(s) would still be carrying traffic. )

Minimises the end to end delay by giving VoIP packets absolute priority without sacrificing bandwidth to do so (traditional methods of prioritising voice throw away 20-30% of the available link capacity in order function effectively, and in any case often the end user has no control over how, or even if, this happens.)

Minimises jitter by ensuring that if a voice packet needs to be sent, it can be without having to wait for any data packets ahead of it.

Minimises packet loss by controlling congestion along its path. In addition, if there is fundamental loss within the network not caused by user traffic, it provides tools to spot this immediately and to counteract it by sending redundant data if needed.

Minimises overheads associated with voice by stripping out those overheads whilst the packets are carried within the tunnel, and restoring them on exit.

Provides real time information about the quality of the link. This isn't obtained by using simple pings, which can be misleading at best, but by analysing your actual data as it passes across the network.

These features make a huge difference to the viability of voice on any network, as you'll see from the next section.



## TEST RESULTS

In a traditional test results section, we'd present side by side comparisons of a bare network, and then one with Aritari's ViBE SD-WAN over the top, so that you'd instantly see how much of an improvement there was. In this case though, that's completely pointless - the capabilities of a ViBE enabled link when used for carrying voice and data traffic are simply an order of magnitude greater than one without, so instead we present the results in separate subsections, with the Aritari network pushed more to its limits.

What these results show unequivocally is that regardless of how good you think your network is, if you're running it for voice and data then you're taking a risk with quality!

### Unaccelerated Network

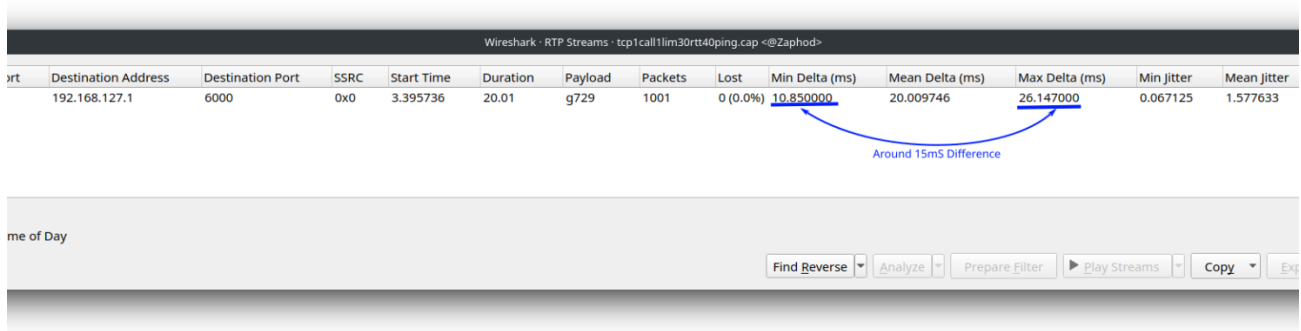
For the bare network tests, we simulated a 10 megabit symmetrical ( the same speed up and down ) link with a 40mS round trip time ( i.e. 20mS in each direction. ) We then used a call generator to make calls, and accessed a web server at the same time to download some files ( we actually used a tool called **wget**. ) For each test, the traffic was captured and then analysed using the Wireshark tool, which generated the results you see here.

When simulating the network, we've kept the router packet buffers small. As explained in part 1, every router connected to a network will have a packet store ( or buffer ) which can hold a number of data packets waiting for space to be sent on the network. If these buffers are large, then packet loss is reduced ( because even if the network is busy, additional packets just queue up waiting until they can be sent ) but at the expense of creating a delay, so normally routers are set with small buffers. This takes advantage of the fact that TCP ( used for most file transfers and web page access, for example ) will see packet loss and slow down its sending rate so as not to create congestion in the network. It also means that any additional latency caused by a busy network is kept to a minimum, which is important for voice.

Note that all of these tests are using the G.729 CODEC at 20mS, so we're trying to make it fairly easy for the network to cope with!

## A Single TCP Stream and a Single Voice Call

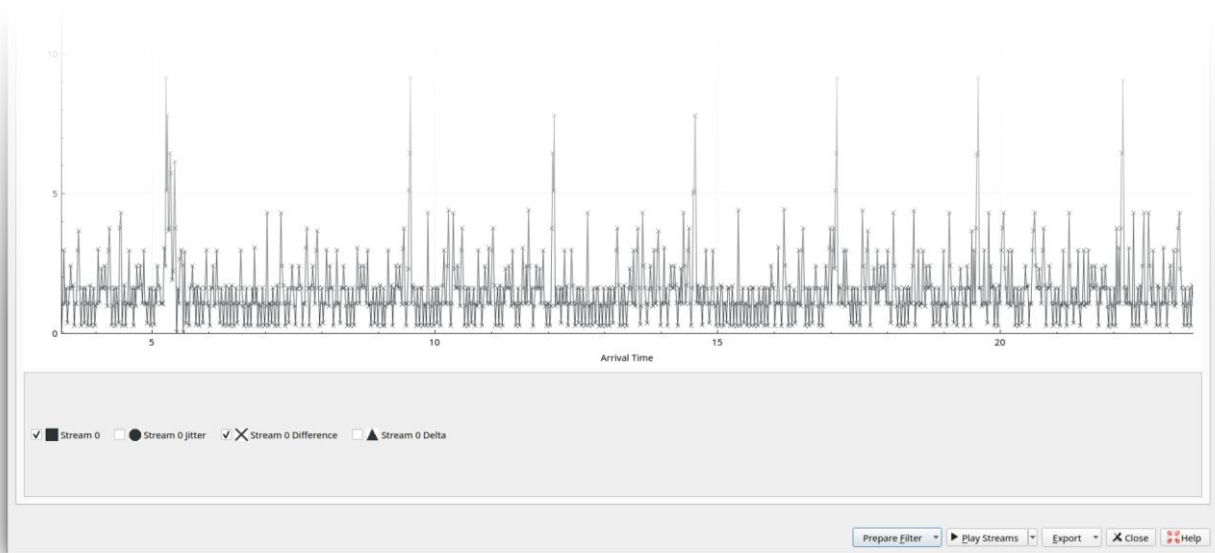
You would hope that a 10mbit/s link could cope with just one file being downloaded and one voice call - after all, the voice is only 8 kbit/s! You'd be right... at least mostly. Here's Wireshark's overview of the voice stream:



Port	Destination Address	Destination Port	SSRC	Start Time	Duration	Payload	Packets	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter
8000	192.168.127.1	6000	0x0	3.395736	20.01	g729	1001	0 (0.0%)	10.850000	20.009746	26.147000	0.067125	1.577633

Around 15ms Difference

As you can see from the "Lost" column, the call didn't lose any packets and things look fairly ok - the only real point to notice is that the "Max Delta" ( which is the time between one packet and the next ) is just over 26mS and the minimum is just under 11mS, whereas for our stream the ideal value is 20mS. In practice this is simply converted into an additional delay of maybe 15mS by the receiving end ( because it has to cope with both extremes, ) which is not going to be noticeable. ( Note that the "jitter" figure calculated by Wireshark is smoothed so not really representative of what the receiver has to cope with. ) Just to see how these "delta" values are across the call, here's a graph of how the gap between each packet differs from the 20mS ideal figure.

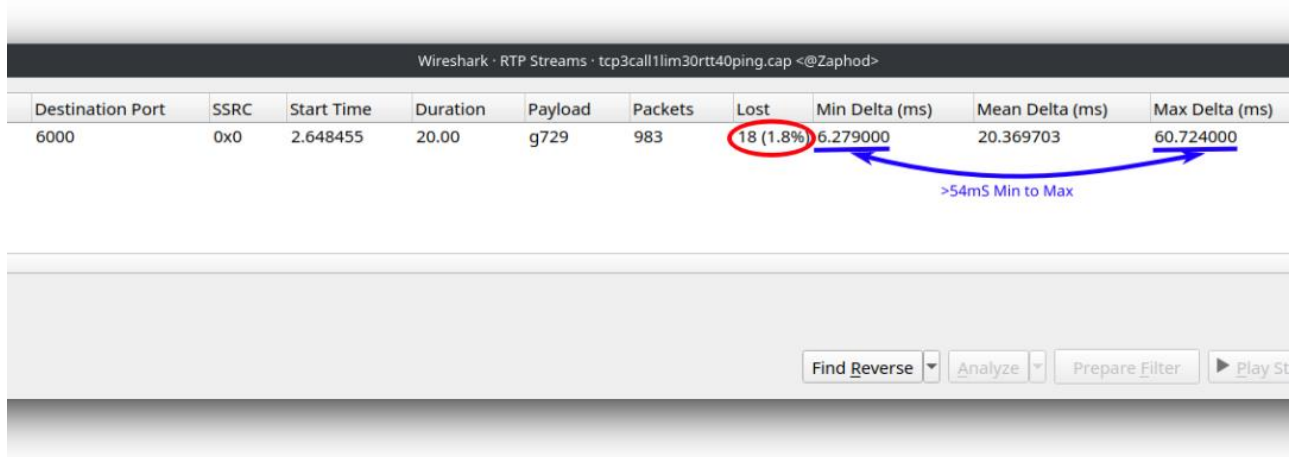


Whilst this isn't disastrous by any means, you can already see the effects on the voice stream of even having a single file transfer going on, and a continuous one at that. In a real network, there will almost certainly be more happening than that - web pages often contain 10's of individual images, for example, and often they'll be downloaded at the same time.

There also won't just be a constant set of data for the whole duration of the call - rather transfers will be continuously starting and stopping.

### Three TCP Streams and a Single Call

OK, let's up the ante slightly and introduce an additional two file transfers into the mix. As in the previous case, the transfers were started just after the call was, but otherwise were then continuous for the duration of the call. Here's what Wireshark has to say about the voice call now:

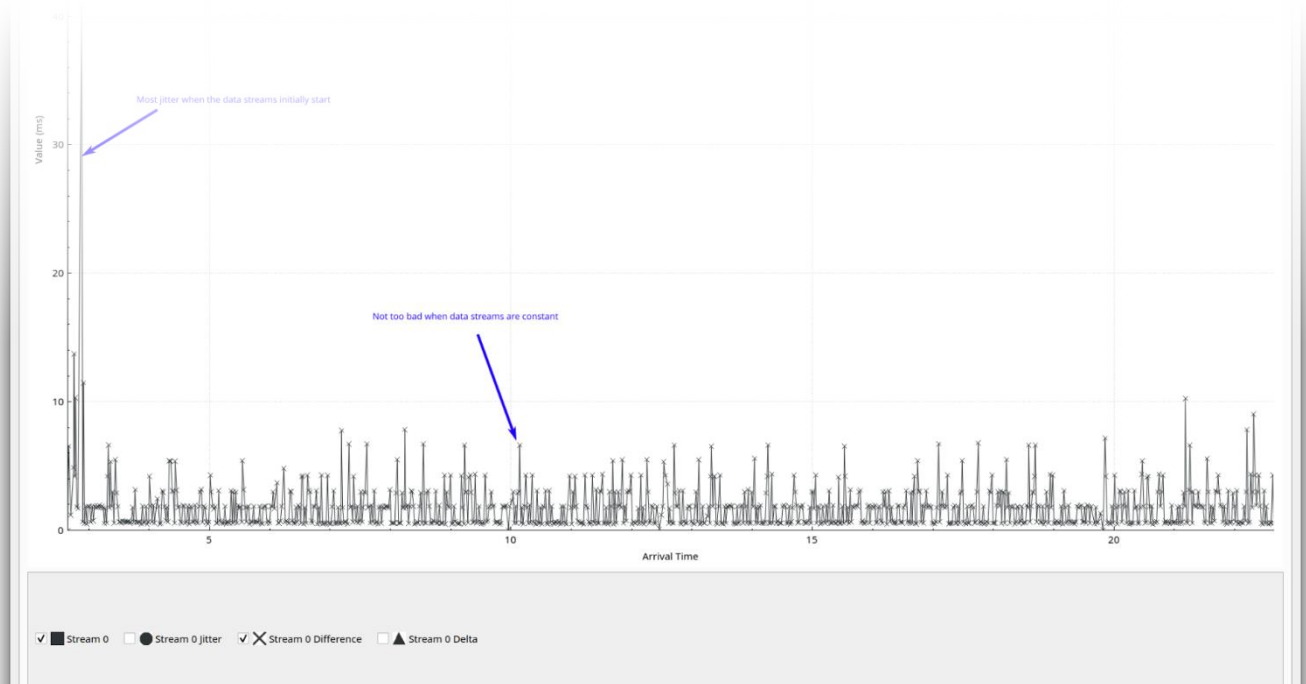


Destination Port	SSRC	Start Time	Duration	Payload	Packets	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)
6000	0x0	2.648455	20.00	g729	983	18 (1.8%)	6.279000	20.369703	60.724000

>54mS Min to Max

Wait... what? Here we have a 10 megabit link carrying a single 8 kbit voice call and three file transfers ( TCP streams ) and the call has 1.8% packet loss... how can that be? Unfortunately, this is the reality of the mixed voice and data network. In actual fact, even here it's not that likely that the loss will be particularly noticeable on the call - the quality will suffer, but will only be a "bit worse" than it would be without. Arguably of greater concern is the fact that the time between the minimum and maximum delta values is now over 50mS, so the receiving end will need to add a 54mS additional delay in order to play back the stream at the required, constant, rate - so we have a 20mS delay to encode the voice, plus a 20mS one way delay, then an additional 54mS jitter buffer delay, giving close to 100mS end to end delay as an *absolute minimum*. It wouldn't take much for this delay to start being unacceptable, especially if the call itself has to travel a long distance ( 20mS is only our simulated "last mile" link. ) For example, there are many CRM/call management solutions that are based on Twilio's PoP network, such as Freshworks, and a popular option with these is the use of "Bring Your Own Carrier," where the customer isn't tied to a particular telephony company to supply the telephone lines and numbers. However, in this setup, calls already have to travel from the customer site, to the nearest Twilio PoP, and back to the carrier - if there isn't a *very* local Twilio PoP available then it's unlikely to be possible to use any system using this configuration and maintain call quality ( because by definition, even a "dedicated" link needs to carry both voice and data to the PoP, and so will have these issues, meaning that the underlying delay needs to be very small to start with so that jitter doesn't problems. ) For example, the nearest Twilio PoP for South Africa is in Europe!

Let's look at the difference graph as before:



This graph sheds a little more light on the issue - you can see that most of the problem was caused at the start of the call, and it's not a coincidence that this is exactly when the download streams were started. At first glance, then, this isn't quite as bad as it first appeared, though you have to remember a couple of things:

1. The receiver needs to cope with the difference in the spacing of the packets, regardless of when that difference happens, since it has to play back the voice at a constant rate. Therefore the additional jitter buffer delay will still be the same even though the major delay was at the start ( technically, it's possible to very slowly shrink the buffer by playing back at an imperceptibly faster rate for a while, but..... )
2. This was the result of three TCP streams starting shortly after the voice call started and then continuing for the duration. In a real network, TCP ( and other ) streams will be starting and stopping all of the time.

## What IS Going on Here?

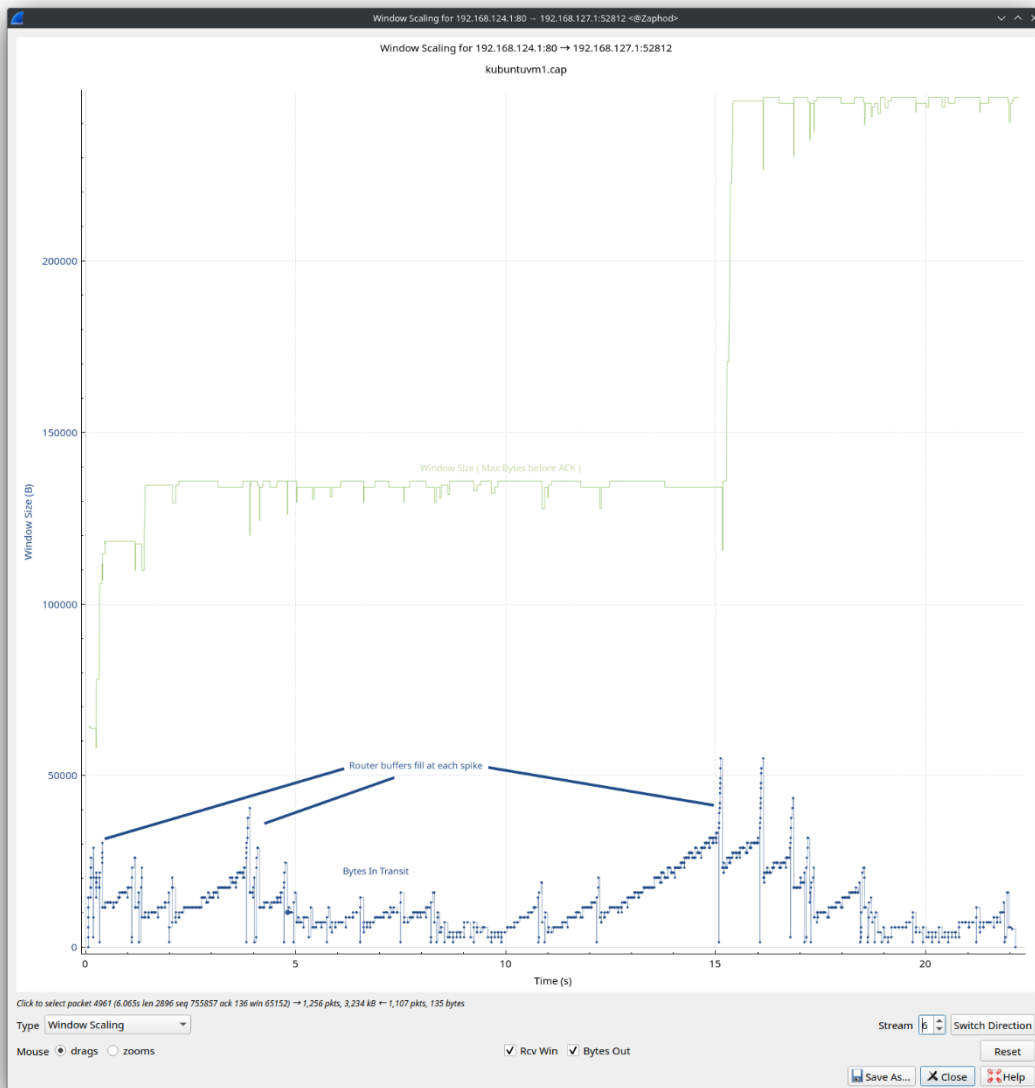
Whilst we could show other combinations of more calls and more streams, the reality is that what's here is really the baseline of what to expect in a real network, and adding more traffic simply makes things worse. The question, though, is why? Surely a 10 megabit network should cope with this fairly low level of traffic?

The answers actually lie in part 1 of this series about the workings and failings of TCP. As explained there, TCP tries to work out how fast it can send data across the network, and so has built in algorithms ( several alternative ones, actually ) which slow down the speed at which packets are sent so that network congestion doesn't become an issue. However, when a TCP session starts, it has *no idea* what conditions it faces, so historically it started very slowly and accelerated until it experienced loss. Modern day TCP implementations still do this, but since networks have generally got faster, TCP starts off assuming that the network is faster. The other issue is that it *has* to cause congestion on the network, however briefly, in order to work out where it should back off, and it *keeps* doing this just in case more bandwidth becomes available.

Consider our 10 megabit network for a moment, with 40ms round trip time. With overheads involved, a 10 megabit link can transfer around 1 megabyte of data every second, give or take. Since ours has a 40ms RTT, that means that TCP *should* send 1 MB x 0.04 of data before expecting an acknowledgement from the other side... however it's unlikely that will be the case ( because initially it has no idea. ) Since the sender will probably be connected directly to a fast link ( maybe 1Gbit or even 10Gbit Ethernet ) all data up until the amount it decides to send initially ( the "initial window size" ) will be sent as quickly as it can, so when it hits our slower link, the router buffers will fill. If you're lucky, this initial amount of data won't completely fill the router buffers, because once the buffers are full, your poor voice call has nowhere to go, so packets are dropped. The generally accepted standard these days is to use an initial window size of 10 packets, which normally is around 15 Kbytes - our link has buffers with a depth of 30 packets ( though they're also used to create the link delay) so that's why a single TCP stream caused no loss for the call.

However, what happens when there's already some other traffic on the network? Well, that other traffic, assuming it's also TCP, will already be holding the router buffers at a certain level, in order that the throughput is as fast as it can be. Any new TCP stream will do the same as the first, but this time with a set of buffers which in effect is smaller... it's no coincidence that in our case, 3 TCP streams causes loss for the voice call.

Let's see this in action:



We started 10 simultaneous TCP streams and this graph is Wireshark's analysis of the assumed window size and the number of "bytes on the wire" ( the amount of data that's been sent but not yet acknowledged ) of the sixth of those streams, so the link is already being fully utilised when this one starts. The blue line is the interesting one, because you can see that the amount of data sent instantly jumps to around the 15K mark ( as we predicted ) and then settles at around that level. However, you can see that during the transfer, there are jumps every now and then where the algorithm is probing to see if there is more space available on the link ( so it suddenly sends a lot more data, which has to go somewhere, and again fills the router buffers. )

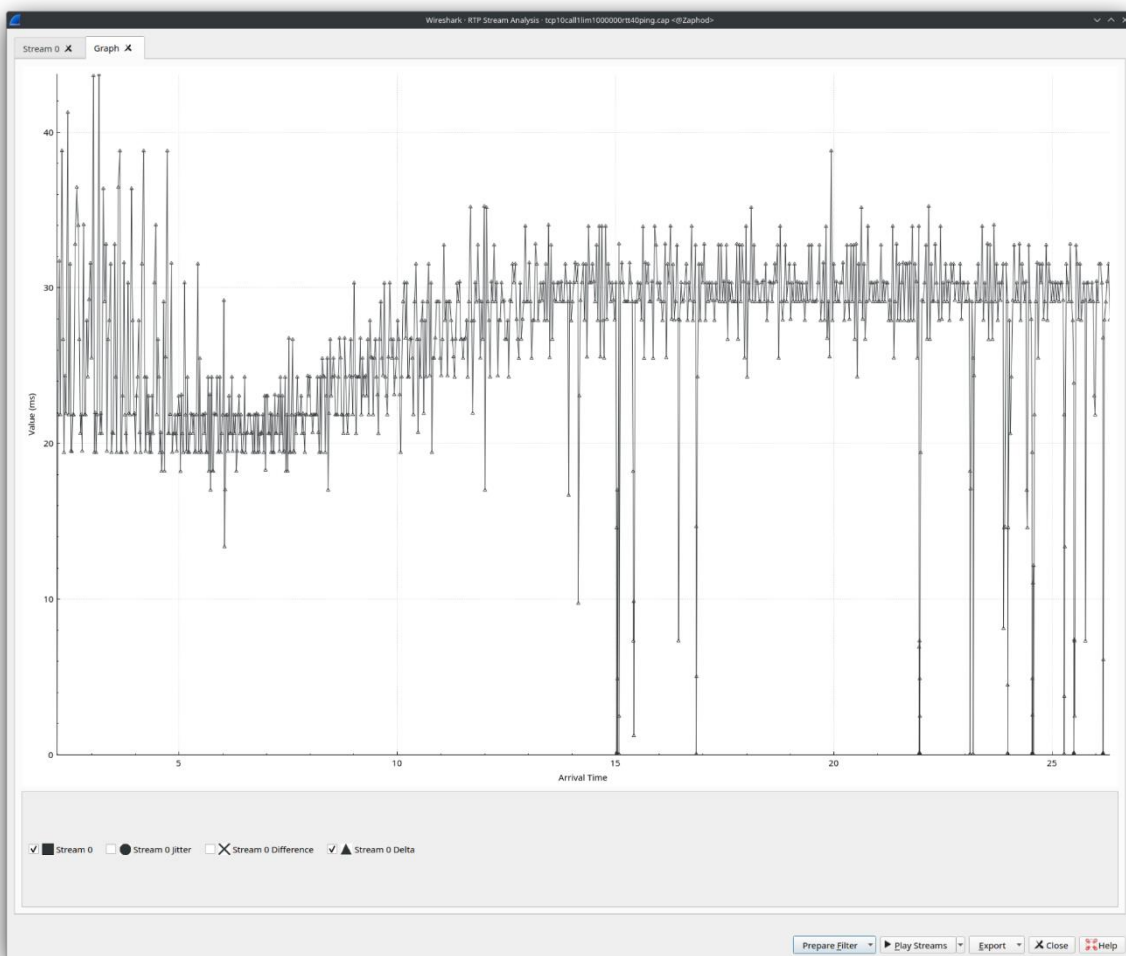
## How Big a Buffer is Needed?

Increasing the router buffer size will ( generally speaking) increase the number of TCP streams that can be carried without causing packet loss for other users of the network.

However, the downside is that more queued packets means more of a delay before packets at the back of the queue can be sent.

We did an experiment to see just how big buffers needed to be in order to support 10 TCP streams without causing packet loss on our 10 megabit link, and the results are quite shocking!

Here's a graph of the time gap between successive packets for our call, with 10 TCP streams started just after the start of the call, each stream starting 0.1s after the previous one:



This shows that once the TCP streams had started, our voice packets were received more or less 30mS apart, so in effect the router buffer just kept filling slowly and by the end of the call packets were being received 4 seconds after they were sent. The answer to the question of “how big,” therefore, is “infinite,” since the TCP algorithm used in this particular case (by Ubuntu Linux) is confused by the fact that it’s not seeing loss anymore, so always tries to send data faster than it should.

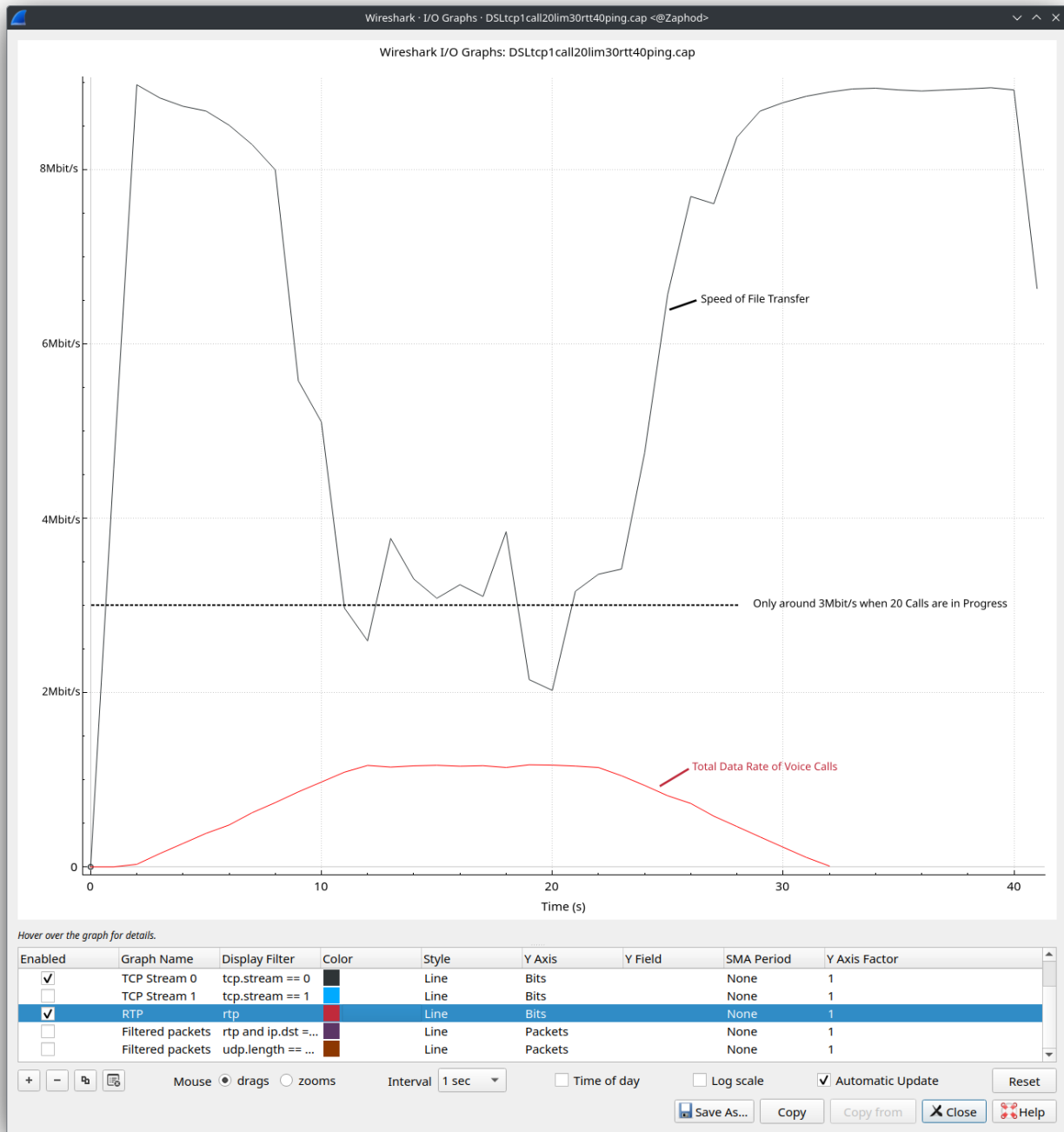
## Mitigations Without ViBE

A mechanism exists that serves to help reduce the issues experienced by voice traffic (or any other data that needs special priority or treatment) called “Quality of Service,” or QoS. This is a blanket term used to describe, generally, the use of multiple packet “queues” within a router, each of which stores packets from different streams and/or different types of traffic. For example, there may be a queue for voice packets, and a queue for other forms of traffic. Whenever a packet needs to be sent onto the network, the voice queue gets looked at first, and if there are any packets there they will get sent before any others, meaning that these voice packets don’t experience the packet loss caused by TCP probing the network. This can work well when it can be implemented, however:

- It has to be configured on both ends of the link, and on the routers that are connected to the slowest link in the chain between the source and destination for packets. Normally, the slowest link is the “last mile” to the end user, so that’s at least achievable... however the end user invariably has no control over the routers, and certainly not the router at the ISP end of their link.
- Voice traffic is quite difficult for routers to identify, so deciding which packets need special treatment is impossible unless the source and destination addresses or some other characteristic are known. Often the ISP isn’t the provider of the voice services, in which case it’s not really viable to configure QoS in any meaningful way.
- Even if you can configure QoS on a link, there is, as ever, a tradeoff. Realistically, when QoS is in use, only around 70% of the original bandwidth of the link can be used, otherwise it starts to become ineffective. Most people paying for a link want to be able to use all of it!

## Testing Efficiency

We touched earlier on the fact that VoIP is very inefficient because of the small packet sizes used, especially when carried over a DSL or other ATM based link ( which many private circuits still are. ) In order to show this, we’ve again simulated a 10 megabit symmetrical link, but this time carried over an ATM protocol. We started a single file transfer, and then started 20 G.729 voice calls, to see what the effect on throughput would be. Here’s the resulting graph:



Throughput of a Single File Transfer and 20 Voice Calls on a 10Mbit/s Link without VIBE

This graph shows the data transfer rate of the file transfer ( TCP stream ) and that of the voice calls ( RTP ) which were started at half second intervals for 20 seconds each ( so 10 seconds at maximum calls on the line.)

You can see that the TCP stream starts off quite happily at around 9 mbit/s but quickly deteriorates to an average of more like 3 mbit/s with all calls in progress, and you can see that the algorithm is really struggling at that level. The voice calls were "ok," which is to say there was no loss that would be particularly noticeable, though if we'd started the TCP stream once all calls were in place this would have caused a spike in loss ( and hence voice problems ) as the stream started. We also tried 30 voice calls but these calls were completely unacceptable. Based on the earlier discussion about ATM this is perfectly logical - the graph above doesn't include overheads, and we know that G.729 at 20mS takes over 5 times the bandwidth that it should, so the portion of the link used by our 20 calls is more like 6 megabits rather than just over 1 as shown. The uninitiated would assume that 30 x 8kbit/s calls was easily achievable on a 10 megabit link, but hopefully you now know better!

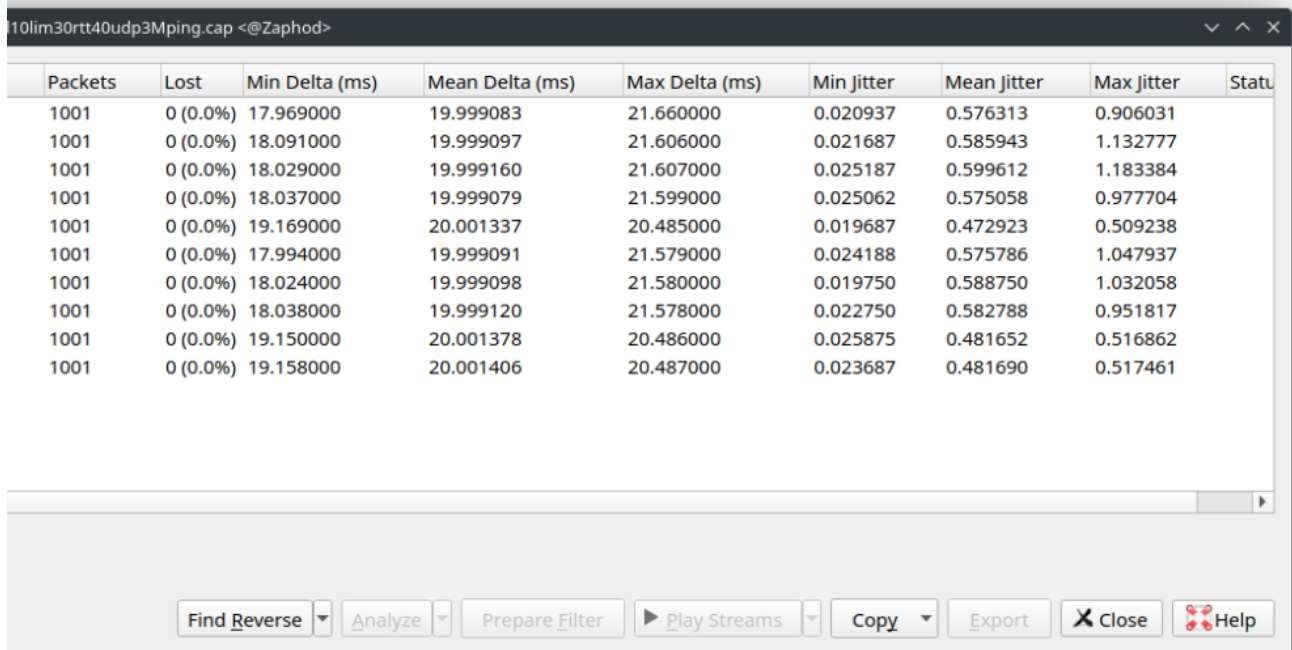
## Aritari's Solution

The Aritari ViBE solution can be deployed as a simple point to point VPN, or as a fully fledged SD-WAN, running over existing infrastructure. It can also be supplied as a managed service ( where a third party manages the configuration ) or under custom control at customer premises. In order to see the difference, we started a ViBE tunnel over the same networks used above with no changes, to see what happened.

Rather than step through all of the various scenarios, we'll just present a small selection of slightly more demanding tests.

### 10 Voice Calls, 10 Downloads, 3 Megabit UDP Stream

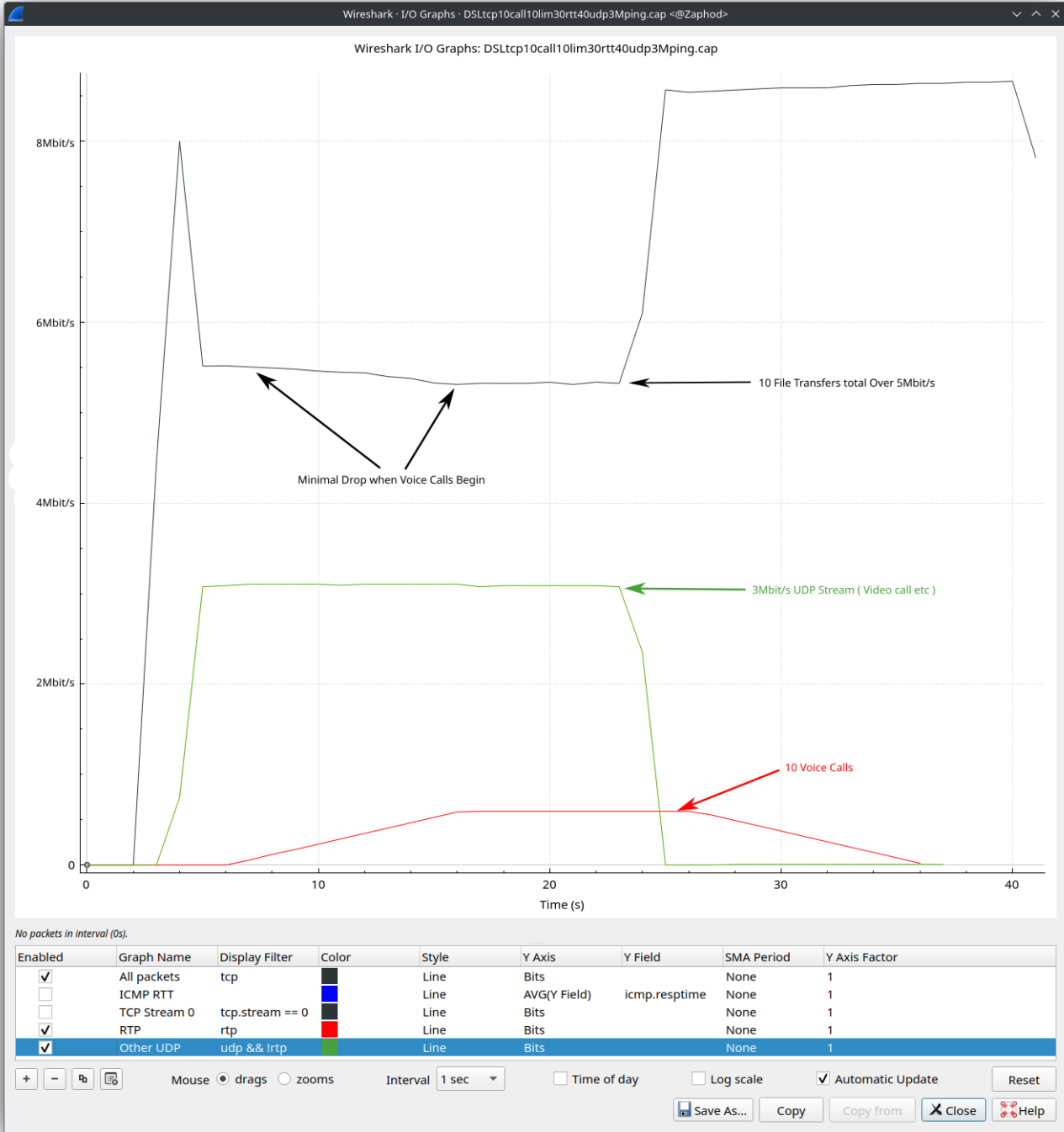
Over the simple 10 megabit network we decided to run this combination of traffic - the additional UDP stream just takes a constant 3 megabit of the bandwidth and might be a video conference or similar. Here's the analysis of the streams themselves:



Packets	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Statu
1001	0 (0.0%)	17.969000	19.999083	21.660000	0.020937	0.576313	0.906031	
1001	0 (0.0%)	18.091000	19.999097	21.606000	0.021687	0.585943	1.132777	
1001	0 (0.0%)	18.029000	19.999160	21.607000	0.025187	0.599612	1.183384	
1001	0 (0.0%)	18.037000	19.999079	21.599000	0.025062	0.575058	0.977704	
1001	0 (0.0%)	19.169000	20.001337	20.485000	0.019687	0.472923	0.509238	
1001	0 (0.0%)	17.994000	19.999091	21.579000	0.024188	0.575786	1.047937	
1001	0 (0.0%)	18.024000	19.999098	21.580000	0.019750	0.588750	1.032058	
1001	0 (0.0%)	18.038000	19.999120	21.578000	0.022750	0.582788	0.951817	
1001	0 (0.0%)	19.150000	20.001378	20.486000	0.025875	0.481652	0.516862	
1001	0 (0.0%)	19.158000	20.001406	20.487000	0.023687	0.481690	0.517461	

You can see from this that all of the voice streams are about as good as they can be.

Here's the associated throughput graph:

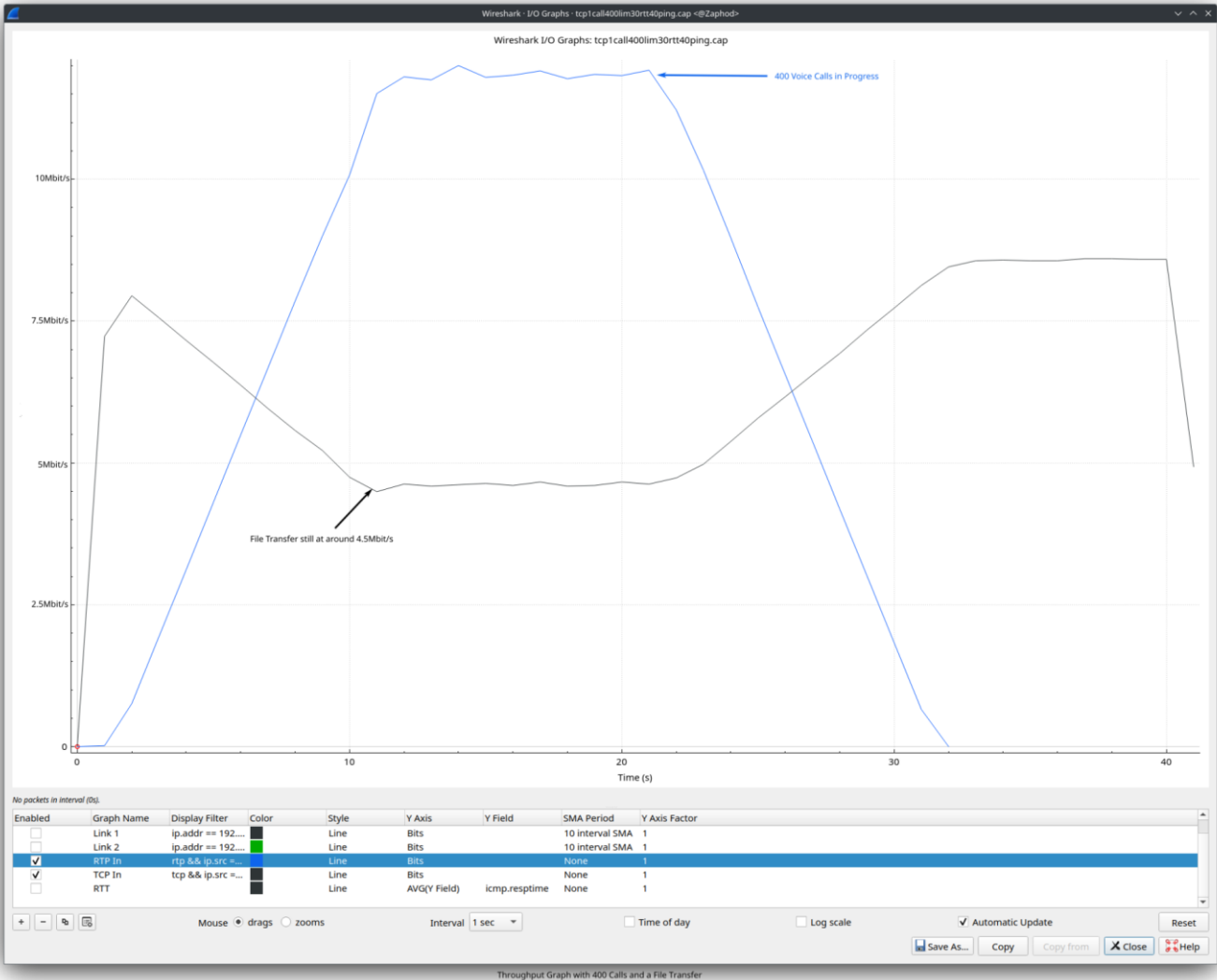


Throughput of 10 Voice Calls + 10 File Transfers + a 3Mbit/s Video Stream on an Aritari VIBE Network

As you can see, each component ( multiple TCP streams, the UDP and RTP streams ) happily take up their needed space on the link, and play very nicely together. You can also see that when the 10 voice calls ( red line ) begin, the data transfer rate for the file transfers actually drops very little ( in keeping with only a little over 8 Kbit/s being used per call ) which is not the case without using ViBE.

## DSL/ATM Test

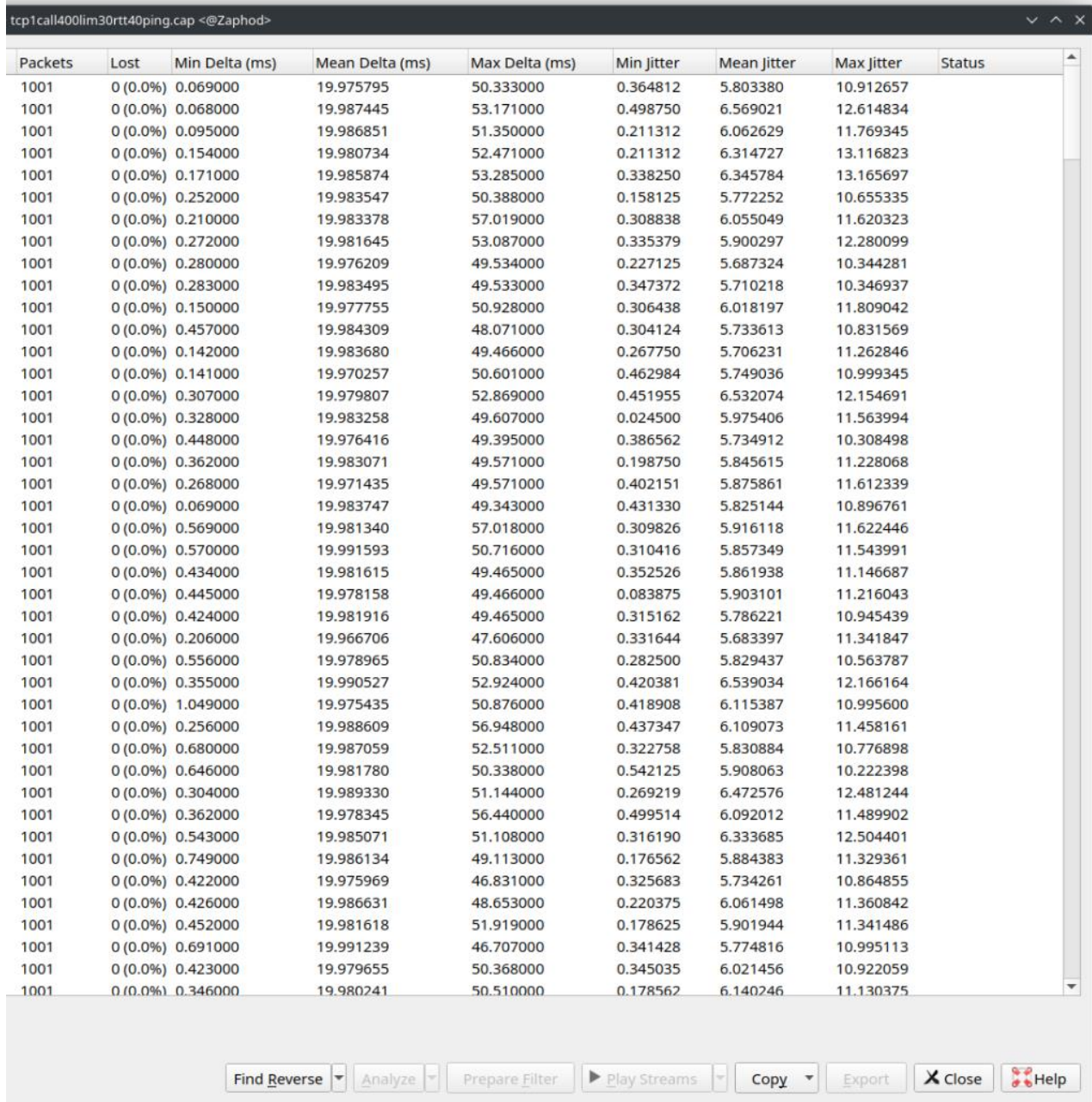
For this test we decided to ramp it up a little... here's the throughput graph:



Bear in mind that in order to show an extreme case, we're really pushing our test hardware to the limit here ( rather than the network - that could easily take more if we'd used a bigger box! ) This is why the RTP line isn't flat at the top - indicating that there's at least some jitter there. The other thing to notice is that the RTP graph actually goes well above 10 megabits even though the link is less than that - this is because although wireshark isn't showing the ATM/DSL overheads, it *IS* showing the other VoIP overheads, which ViBE strips out before sending through its tunnel ( and adds back again at the other end. ) It's important to understand that we're NOT changing the voice payload in any way, so the quality of the calls is unaffected by this process.

Here's Wireshark's analysis of ( a snapshot of ) these streams:

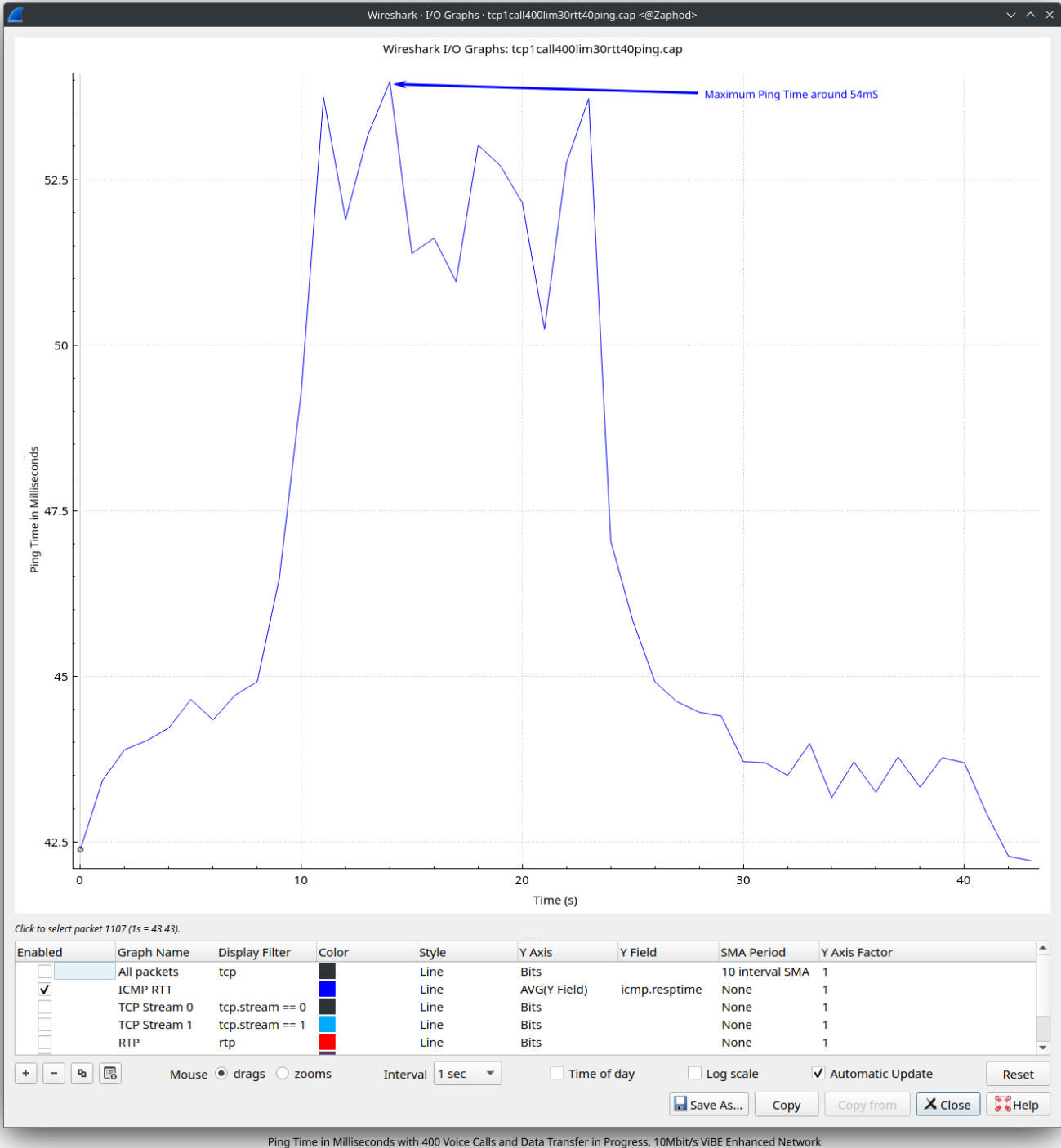
You can see here that the min to max delta is around 50mS, though given that the mean is close to the ideal 20mS most of the time everything is still perfect, but because we're pushing the hardware to the limit here every now and again the processor is overloaded and a small delay is introduced.



Packets	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Status
1001	0 (0.0%)	0.069000	19.975795	50.333000	0.364812	5.803380	10.912657	
1001	0 (0.0%)	0.068000	19.987445	53.171000	0.498750	6.569021	12.614834	
1001	0 (0.0%)	0.095000	19.986851	51.350000	0.211312	6.062629	11.769345	
1001	0 (0.0%)	0.154000	19.980734	52.471000	0.211312	6.314727	13.116823	
1001	0 (0.0%)	0.171000	19.985874	53.285000	0.338250	6.345784	13.165697	
1001	0 (0.0%)	0.252000	19.983547	50.388000	0.158125	5.772252	10.655335	
1001	0 (0.0%)	0.210000	19.983378	57.019000	0.308838	6.055049	11.620323	
1001	0 (0.0%)	0.272000	19.981645	53.087000	0.335379	5.900297	12.280099	
1001	0 (0.0%)	0.280000	19.976209	49.534000	0.227125	5.687324	10.344281	
1001	0 (0.0%)	0.283000	19.983495	49.533000	0.347372	5.710218	10.346937	
1001	0 (0.0%)	0.150000	19.977755	50.928000	0.306438	6.018197	11.809042	
1001	0 (0.0%)	0.457000	19.984309	48.071000	0.304124	5.733613	10.831569	
1001	0 (0.0%)	0.142000	19.983680	49.466000	0.267750	5.706231	11.262846	
1001	0 (0.0%)	0.141000	19.970257	50.601000	0.462984	5.749036	10.999345	
1001	0 (0.0%)	0.307000	19.979807	52.869000	0.451955	6.532074	12.154691	
1001	0 (0.0%)	0.328000	19.983258	49.607000	0.024500	5.975406	11.563994	
1001	0 (0.0%)	0.448000	19.976416	49.395000	0.386562	5.734912	10.308498	
1001	0 (0.0%)	0.362000	19.983071	49.571000	0.198750	5.845615	11.228068	
1001	0 (0.0%)	0.268000	19.971435	49.571000	0.402151	5.875861	11.612339	
1001	0 (0.0%)	0.069000	19.983747	49.343000	0.431330	5.825144	10.896761	
1001	0 (0.0%)	0.569000	19.981340	57.018000	0.309826	5.916118	11.622446	
1001	0 (0.0%)	0.570000	19.991593	50.716000	0.310416	5.857349	11.543991	
1001	0 (0.0%)	0.434000	19.981615	49.465000	0.352526	5.861938	11.146687	
1001	0 (0.0%)	0.445000	19.978158	49.466000	0.083875	5.903101	11.216043	
1001	0 (0.0%)	0.424000	19.981916	49.465000	0.315162	5.786221	10.945439	
1001	0 (0.0%)	0.206000	19.966706	47.606000	0.331644	5.683397	11.341847	
1001	0 (0.0%)	0.556000	19.978965	50.834000	0.282500	5.829437	10.563787	
1001	0 (0.0%)	0.355000	19.990527	52.924000	0.420381	6.539034	12.166164	
1001	0 (0.0%)	1.049000	19.975435	50.876000	0.418908	6.115387	10.995600	
1001	0 (0.0%)	0.256000	19.988609	56.948000	0.437347	6.109073	11.458161	
1001	0 (0.0%)	0.680000	19.987059	52.511000	0.322758	5.830884	10.776898	
1001	0 (0.0%)	0.646000	19.981780	50.338000	0.542125	5.908063	10.222398	
1001	0 (0.0%)	0.304000	19.989330	51.144000	0.269219	6.472576	12.481244	
1001	0 (0.0%)	0.362000	19.978345	56.440000	0.499514	6.092012	11.489902	
1001	0 (0.0%)	0.543000	19.985071	51.108000	0.316190	6.333685	12.504401	
1001	0 (0.0%)	0.749000	19.986134	49.113000	0.176562	5.884383	11.329361	
1001	0 (0.0%)	0.422000	19.975969	46.831000	0.325683	5.734261	10.864855	
1001	0 (0.0%)	0.426000	19.986631	48.653000	0.220375	6.061498	11.360842	
1001	0 (0.0%)	0.452000	19.981618	51.919000	0.178625	5.901944	11.341486	
1001	0 (0.0%)	0.691000	19.991239	46.707000	0.341428	5.774816	10.995113	
1001	0 (0.0%)	0.423000	19.979655	50.368000	0.345035	6.021456	10.922059	
1001	0 (0.0%)	0.346000	19.980241	50.510000	0.178562	6.140246	11.130375	

If you're paying attention you might have noticed WHY the hardware may be a little taxed - as the title indicates, the throughput graph and stream list are the result of 400 simultaneous calls being carried on a link which previously couldn't manage 30. As the TCP trace on the throughput graph indicates, there's still plenty of capacity for data at that level.

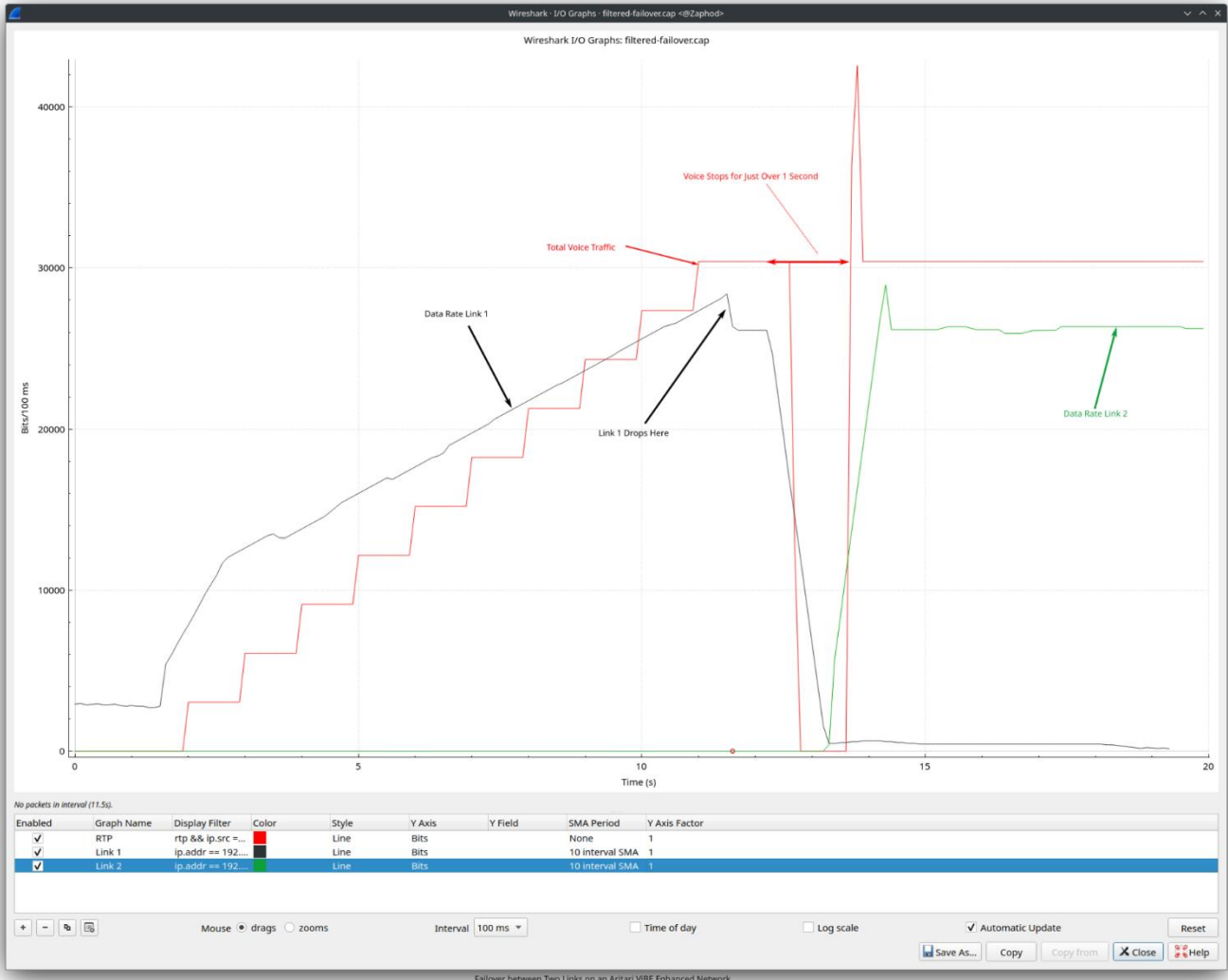
Just to finish off this section, we had a “ping” running across the link during this test, and here’s the round trip time graph for that:



Even with the links being pushed this way, the ping time only peaks at around 54ms.

## Redundancy and Failover

In the non-Aritari enhanced network case, we didn't even touch on what happens when a link fails. This is because there's no easy way of adding failover to a bare link - you always need some sort of device or scripting to control such a link, and anything we put together probably wouldn't be representative of what you have available to you. However, since ViBE handles failover as part of its core functionality, we *can* show you how *we* do it.



The above graph shows three traces:

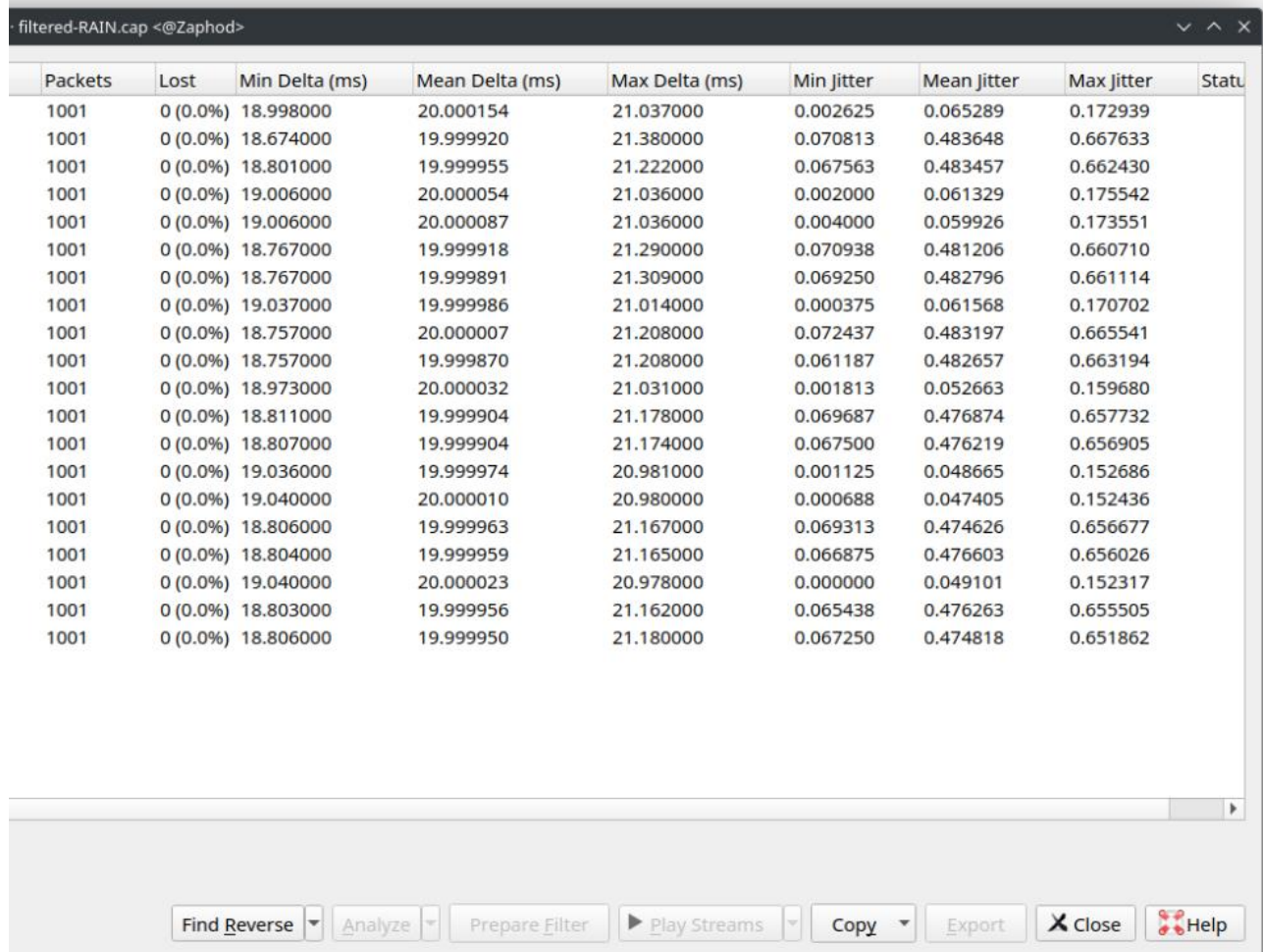
- The red line shows the overall RTP transfer rate, which consists of 20 calls which were started two at a time a second apart.
- The black line shows the traffic on the primary link. You can see that we disconnected the link just as the final call was connected.
- The green line shows the traffic on the failover link.

As you can see from this, ViBE manages to bring up the backup link and restore RTP traffic just over a second after the main link fails. These are using the default configuration - it's actually possible to configure faster failover than that if the network is good enough. No changes to the voice calls happen after the failover, which is why they all continue immediately once the redundant link is in service.

For businesses that cannot tolerate downtime at all, we have a mode that we call RAIN ( Redundant Array of Inexpensive Networks ) which duplicates traffic over more than one link. Here's the equivalent graph in that mode:



You can see here that the calls are completely unaffected by the link failure - here's the Wireshark call analysis just for confirmation:



Packets	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter	Mean Jitter	Max Jitter	Statu
1001	0 (0.0%)	18.998000	20.000154	21.037000	0.002625	0.065289	0.172939	
1001	0 (0.0%)	18.674000	19.999920	21.380000	0.070813	0.483648	0.667633	
1001	0 (0.0%)	18.801000	19.999955	21.222000	0.067563	0.483457	0.662430	
1001	0 (0.0%)	19.006000	20.000054	21.036000	0.002000	0.061329	0.175542	
1001	0 (0.0%)	19.006000	20.000087	21.036000	0.004000	0.059926	0.173551	
1001	0 (0.0%)	18.767000	19.999918	21.290000	0.070938	0.481206	0.660710	
1001	0 (0.0%)	18.767000	19.999891	21.309000	0.069250	0.482796	0.661114	
1001	0 (0.0%)	19.037000	19.999986	21.014000	0.000375	0.061568	0.170702	
1001	0 (0.0%)	18.757000	20.000007	21.208000	0.072437	0.483197	0.665541	
1001	0 (0.0%)	18.757000	19.999870	21.208000	0.061187	0.482657	0.663194	
1001	0 (0.0%)	18.973000	20.000032	21.031000	0.001813	0.052663	0.159680	
1001	0 (0.0%)	18.811000	19.999904	21.178000	0.069687	0.476874	0.657732	
1001	0 (0.0%)	18.807000	19.999904	21.174000	0.067500	0.476219	0.656905	
1001	0 (0.0%)	19.036000	19.999974	20.981000	0.001125	0.048665	0.152686	
1001	0 (0.0%)	19.040000	20.000010	20.980000	0.000688	0.047405	0.152436	
1001	0 (0.0%)	18.806000	19.999963	21.167000	0.069313	0.474626	0.656677	
1001	0 (0.0%)	18.804000	19.999959	21.165000	0.066875	0.476603	0.656026	
1001	0 (0.0%)	19.040000	20.000023	20.978000	0.000000	0.049101	0.152317	
1001	0 (0.0%)	18.803000	19.999956	21.162000	0.065438	0.476263	0.655505	
1001	0 (0.0%)	18.806000	19.999950	21.180000	0.067250	0.474818	0.651862	

RAIN mode doesn't only help in situations where a link completely fails, but also allows you, for example, to combine two links which individually have high packet loss and jitter, but when used as a RAIN pair become perfectly acceptable ( since the Aritari ViBE system will use the best link of the two on a packet by packet basis.) In fact, RAIN mode can even be used on a single link, so that links with high random packet loss also become very usable without the need to have multiple circuits.



## SUMMARY

Hopefully this has been an interesting introduction to how a voice network differs greatly from one designed for data, and why it can be so difficult to run voice and data on the same link. We've shown how Aritari's ViBE SD-WAN/VPN solution allows voice and data to coexist in a way that doesn't compromise either the calls or the file transfer performance.

Without ViBE, very few calls can be carried with a high degree of certainty of quality, even on relatively high bandwidth links, especially when other data is present - which is always true when CRM type systems are in use even when the links are supposedly "dedicated for voice."

In addition, we hope that we've made you think about the resiliency of your network, and what might happen if one of your links fails, or suffers some form of attack - often this aspect isn't considered until it's too late!