ARITARI

NEW WORLD NETWORKS

ARITARI TCP ACCELERATION

WHY MORE BANDWIDTH IS NOT ALWAYS THE ANSWER.

TABLE OF CONTENTS

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 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. Initially developed with Voice over IP (VoIP) in mind, over the years it 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.
- 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.

This document will focus on the TCP acceleration features of the Aritari ViBE SD-WAN solution, and you can see some examples of the difference it makes at the end.

BASICS OF TCP

The vast majority of large data transfer on the Internet today takes place over the Transmission Control Protocol (TCP,) which is a reliable transport mechanism that guarantees delivery of data across a network. Unfortunately, TCP was designed in the early 1970's, when the prevailing wide area network technology was vastly different from that of today. Various modifications to the original TCP specification have been made over the years, which have, to some extent, mitigated some of the original design constraints, though these changes have always had to retain backwards compatibility with existing implementations and therefore are optional. This has resulted in vastly different performance depending on the specific optional features used, and how they are applied, so that the performance of a connection between two hosts can be hugely different depending on many factors such as the operating system in use (or even the exact version of that OS,) and the use of various algorithms to deal with issues such as packet loss or link congestion.

For this reason, Aritari developed a TCP optimisation function within its ViBE SD-WAN product, which eliminates these problems across any network where ViBE is in use, by virtue of the fact that a proprietary protocol is used instead. Outside of the SD-WAN tunnel, this process is completely transparent to the hosts and applications that still believe they're using TCP.

This document will touch on some of the issues with TCP, as well as some information about how Aritari's ViBE system eradicates them.

THE FUNCTION OF A RELIABLE TRANSPORT PROTOCOL

TCP is a reliable transport protocol - that is to say that an application can send data using it in the knowledge that all information sent will be received by the other side of the connection without errors, and in the order that it was originally sent. In a perfect world this would be taken care of by the network itself, though in reality that cannot be guaranteed. For one thing, decisions taken by routing devices can result in packets of information being transmitted out of order - maybe because a route changes or because processing within the device itself results in the order difference. Some bonding solutions (not Aritari's) also pay no regard to what constitutes a stream of packets, and so the same stream of information may flow along multiple paths, with the constituent packets arriving at their destination in the order determined by those individual routes.

Packet loss can (and frequently does) also occur, whether because of momentary signal quality issues on radio based networks, or router buffers being full, or a multitude of other reasons. Packet loss is also frequently the result of a congested network, which doesn't necessarily mean an *overloaded* network, only that in the fraction of a second that a packet wants to be transmitted, there is no room to do so, because there's already one being sent, and the router queue is full of other packets waiting to be sent. A traffic graph may well show that the network is only 10% full, but if you could zoom in to the millisecond range, you'd see that in fact the network is 100% full for several mS at a time.

TCP OPERATION BASICS

In very broad terms, TCP will try to cope with issues on the network in the following ways:

- A connection is established using what's known as a 3-way handshake process. This involves the initiator of the connection sending an "I want to connect" packet, followed by the receiver sending back an acknowledgement before actual data transfer can begin. If the latency of the link is high, or if many connections are having to be made (such as downloading a web page with many images) then the time taken for all of these handshakes can be significant, so an optional extension to the original TCP specification allows for a "fast open," which allows data to be sent with the initial connection request packet. However, there are various security concerns around this, and its use is very hit and miss.

- In common with all Internet protocols, a stream of information is broken down into packets of information, with each of these packets containing routing information so that it can be sent to its destination independently of the others. For historical reasons, these packets are normally around 1500 bytes in size, with 40 bytes of that used for routing information and the remaining 1460 bytes containing the actual data. In reality, other protocols in use on the network may reduce the size further… for example PPP links normally only allow 1492-byte packets. In the best (usual) case then, 2.7% of bandwidth is lost to control data, but often more.

- When the receiving side gets a sequence of packets, it will send back an acknowledgement to the sender to indicate that data has been received. The original TCP specification only allowed for in-order acknowledgements, so if a packet is received that doesn't follow on from previous packets, the receiver will wait until it gets the missing packet before sending an acknowledgement for all data so far received. An update to the protocol, called "selective acknowledgements," allows more modern systems to tell the sender that everything else has been received, but this is optional.

- The sender must therefore keep all data that's been sent in a buffer until it *knows* that the receiver has successfully received the data sent so far. Since machines that are sending the data are normally used for other tasks and can be sending many such streams at once, the amount of data that is kept in these buffers is limited to the "window size." Once the transmission window is full, the sender must stop sending further data until there is space to save the new information. All data in the send buffer will need to be kept for at least as long as it takes to send that information to the other side AND receive an acknowledgement back that it has been received ok. The original TCP specification only allowed for 64k of data to be saved as a maximum, though there is another optional extension "window scaling" that allows this value to be multiplied so that up to one gigabyte can be stored.

- Whilst the connection is ongoing, TCP will attempt to mitigate the effects of congestion on the network, in order to prevent problems caused by the fact that any given link may intermittently have more data to transmit than it is physically able to. If ignored, this fact could cause connections to slow down massively due to the sudden increase in delay or packet loss that would happen in this situation, so various "congestion control algorithms" are used by operating systems to prevent this from happening. There are many such algorithms, but each is suited more to specific cases than others, so inevitably these algorithms involve compromise, and their effectiveness can vary widely.

From this description, it should be recognised that the latency (the time taken for a packet to travel from sender to receiver) can have a huge impact on the amount of data needed to be stored, and hence how much data can be sent before the connection needs to be paused.

ARITARI ACCELERATION FUNCTIONS

Initial Connection

Normally, the initial 3-way handshake happens between the client (the PC, laptop etc) making the initial connection, and the server (web/ftp/etc.) from which it intends to receive, or to which it intends to send, data. The time taken for this connection will be affected by several factors:

- The amount of time the server takes to validate the request. Historically, this would have been a significant factor, but nowadays most hosts will respond in fractions of a millisecond.
- The round trip time (RTT) from the client to the server - it will be at least this length of time before the client can start to send data. As noted in the TCP Operation section, there is a recent mechanism called “fast open”, which allows the client to send data with the initial connection request, so for example it could send information about the web page that it wants to receive without waiting to see if the connection is established. However, there are various security concerns around this mechanism and its implementation is hit and miss (it has to be enabled on both client and server).
- Any packet loss that occurs in establishing the connection. If the initial connection request packet happens to go missing between the client and the server, than this will increase the time taken for the connection to be established by an order of magnitude, since at this point the client has no way to know how long it should wait before trying again - depending on the specific operating system used on the client it can easily be several seconds before retrying.

If an Aritari ViBE connection is in place between the client and server, and TCP Acceleration is enabled, then ViBE acts as a proxy for the connection, which is to say that rather than the client establishing a connection with a server, it *actually* connects to the nearest ViBE device - this is completely transparent to the client machine. Normally the nearest ViBE device will be an Aritari CPE on the same network as the client, meaning that the initial connection can be established in fractions of a millisecond, and the client will begin to send data almost immediately. Also, it is much less likely that packet loss will occur on the local area network (LAN) than it is on the wide area network (WAN.)

Across the SD-WAN connection, ViBE uses its own, proprietary protocol in order to make the connection to the server, and this protocol handles packet loss in a much more efficient way than TCP can, meaning that even if the initial connection request goes missing, end to end communication will still be established quickly.

Finally where the connection exits the ViBE network (at the nearest Aritari device to the server,) a standard TCP connection will be made between the exit ViBE node and the server itself. Again this remains completely transparent to the server, which thinks it is communicating with the client (and vice-versa.)

Data Transfer and Window Size

In the TCP Operation Basics section, we introduced the concept of TCP Window Size, which is the amount of information that one end of the connection can send before having to wait for an acknowledgement. In an ideal case, with no packet loss, then the window size needs to be at least the round trip time of the link (in seconds) multiplied by the real throughput between the client and server - if this isn't the case, then the throughput of the connection will be limited by that rather than the speed of the link itself.

In recent years, the TCP Window Scaling option has increased the maximum size allowed from its initial value of 64 kilobytes, though as with other options this has to be supported by both client and server. Even with scaling, the connection will start off with a fairly small window size and slowly increase it as the connection progresses, providing that there is no packet loss in the meantime, which means that the speed of the connection will NOT initially be the speed of the link (assuming a reasonably fast link or high RTT.) Packet loss, however small, can have a profound effect on this scaling mechanism, since in the event of loss the window size is normally reduced, and the ramp-up process begins again from this new, reduced value. Also, whilst the theoretical maximum window size using scaling is 1 gigabyte, operating systems often have their own limit which is lower than this.

Since the Aritari ViBE connection does not use TCP, the issues of window size and the need for TCP selective acknowledgement to be supported by both ends of the connection vanish for the portion of the link carried by the SD-WAN. In an ideal case, the ViBE link will cover the portion of the journey between client and server with the highest latency and that is most likely to suffer packet loss or congestion, and therefore the transmission speed between client and server will always be as fast as possible. The details about how this works are proprietary, but the effect within the tunnel is that:

- There is no "ramp-up" procedure. The TCP connection will operate at full speed regardless of the available bandwidth and round-trip time. This fact alone can improve throughput significantly, especially where the amount of data to be sent is quite small (the TCP ramp-up would never complete) or the latency of the connection is high (the TCP ramp-up takes longer.)
- Packet loss does not adversely affect the throughput, beyond having to re-transmit the lost packets. This is the result of several factors:
    - When loss is detected, the connection is not immediately slowed down by a reduction in window size as would be the case with TCP.
    - The loss of individual packets of data is recognised immediately, without having to guess how long to wait before deciding that they are actually lost.
    - Only those lost packets are retransmitted, and this happens immediately.

This combination of factors can produce connections which have much higher throughput when using Aritari ViBE rather than TCP alone, sometimes an order of magnitude higher. The actual results achieved will depend very much on underlying network conditions, though the key thing here is that the ViBE connection will be consistently good, and hugely less affected by transient changes in the network conditions.

Congestion Control and Packet Loss

Any given connection will have a limit on the amount of data throughput that it can sustain, and the connection between a client and server will be limited by the slowest link in the chain (normally the "last mile" or customer connection.)

In a typical scenario, a client may be connected to a 1 Gbit/s Ethernet, which is then plugged into a router. This router may then have a 20 Mbit/s upstream connection to the Internet, the other end of which connects to a multi-gigabit backbone. If the client PC in London wants to connect to a server in San Francisco, then there will be various links involved forming part of the backbone, finally terminating in a (hopefully) multi-gigabit connection to the server. If the client is using a typical DSL type link, then the reverse direction will be similar, but the last mile connection might be 70 Mbit/s rather than 20.

In terms of link performance, there are two key factors involved here:

- The RTT (round trip time) between London and San Francisco, which will probably be around 145mS. There will in addition to this be a latency added by the last-mile connection because of the relatively low link speed and also the fact that data has to be encoded and decoded to be sent on a typical customer link, which might be 20mS in this case, making the RTT 165mS
- The last-mile link speed, which is 20 Mbit/s to send and 70 Mbit/s to receive - i.e., almost certainly much slower than any other connection in the chain.

Let's assume that there is no inherent packet loss on this link, and that conditions are perfect. When the client makes the initial connection request in this scenario, it will assume that it has 1 Gbit/s of bandwidth available to all connections that it makes, since this is all it can see (it has no idea about the rest of the connection to the server.) This fact doesn't really matter too much if this is the only connection because the connection request is just a single packet - it will arrive at the router and immediately be sent on to the Internet link, because there's nothing else on there, and all is well. By far the biggest factor in this case is the fact that it will be at least 165mS before the connection is established.

What happens, though, when the client wants to upload to the server? The client believes that it can send 1 gigabit per second, because again this is all the information it has, so it will send as much information as it can as quickly as it can (in the case of TCP, this will be limited to the initial window size, which as you will see is very important here.) The router that is connected to the slower Internet link will receive all of these packets from the client and will start to send them to the network, though quite obviously it can send them nowhere near as quickly as they came in, so it will store ("buffer") what it can't send immediately in a queue, waiting until it CAN be sent. However, at this point the router has two choices:

- Save every packet that it can't send in its buffer, waiting until it *can* send them. The problem with this strategy is that anything else trying to send on the Internet link will also have to be added to the end of this buffer, so that there will be an additional latency incurred by these new packets. If each packet is a typical 1500 bytes, then for a 20 Mbit/s upstream link this additional delay will be around 750 microseconds, which might not sound much, but even if the TCP window size is very small (say 64 Kbytes) then this would increase to 33mS, but the likelihood is much longer.
- Limit how many packets are saved, in order to reduce this additional latency on the link. In this case, it has to throw away any packets that it receives when its buffer is full, meaning that those packets are lost (and hence even if the link itself has no packet loss, it is introduced by this process.)

These days most routers are configured with the second of these strategies, using the theory that increased latency is not good because it affects all connections using the link, whereas packet loss can be detected and the sending of data slowed down in order to prevent buffers from filling. TCP uses this idea by the implementation of what are called “congestion control algorithms.” Depending on which algorithm is in use by the operating system, the TCP connection will be slowed down in various ways in response to packet loss, which also explains why, when packet loss happens on a link, the initial reaction of the TCP sender is to reduce the speed at which it sends.

These issues are, of course, exacerbated by the fact that in a real network, there will be many such connections in progress or trying to be established, and any one of those connections could in theory fill the router’s buffer. No TCP congestion control algorithm is perfect - they all have their strengths and weaknesses, and the average user has no idea what algorithm is in use or what effect that may have on the network. Additionally, other protocols such as UDP have no such built-in brake, relying instead on the individual client/server connection to have its own method of congestion control, a fact which can be exploited by various denial of service attacks and suchlike. Also, of course, the same thing happens on the receiving side, with the service providers routers having the buffers and control of how they’re used.

The net result of all of this is that even in perfect networks, there will be times when the Internet link is not being used to its full potential. Even a single new connection could fill a router buffer and cause packet loss to be seen by all connections, and the higher the utilisation of the network, the worse this issue becomes.

An Aritari ViBE connection removes or greatly reduces the impact of these issues. For one thing, simple linear (first in, first out, or FIFO) buffers are never used, so that any new packets can always potentially be sent on the link. Also, ViBE knows exactly how much bandwidth is available, and so can manage access to that bandwidth in a controlled manner, even when protocols such as UDP which don’t have a built-in congestion control mechanism are used. Individual TCP senders will not see packet loss introduced due to overflowing router buffers - instead ViBE controls their send rate by controlling when it sends acknowledgements, which means that there is no longer any reliance on how good (or otherwise) the congestion control algorithm of a particular client machine might be.

Another significant advantage of using ViBE is that there is no trade-off between very small (associated with connection requests, credit card transactions and interactive terminal sessions) and very large packets (associated with file/data transfer.) This means that interactive sessions such as ssh, POS, bank card, or other character-based systems remain completely responsive even when the network is heavily utilised.

Data Overheads and Inefficiency

All Internet data has overheads which cause inefficiency in sending that information across a network. These overheads occur at several layers of a connection:

- **Physical Layer.**

  Any given link, such as an Ethernet cable or a WAN link, will need to have some way of reliably sending information across it. You may believe that Ethernet, for example, is pretty efficient, but in reality, there are several overheads added to every single packet transmitted, amounting to about 20 bytes.

- **Data Link Layer.**

  This is how two hosts ***on the same network*** communicate. In a typical Ethernet implementation, this amounts to 18 bytes, though other technologies can be even less efficient and their encoding schemes complex. A good example of this last point is DSL, which is still based around a protocol called ATM which was developed originally to carry multiple telephone calls on a link. On an ATM link, every packet of information is broken down into fixed 53 byte "cells," each of which has 5 bytes used for carrying signalling information. In addition, each packet has an 8-byte trailer appended. If there isn't enough data left to fill the end of a cell, it will simply be transmitted empty.

- **Network Layer.**

  This layer is responsible for splitting data into smaller packets, and for allowing these packets to be routed along multiple network segments in order to get to where they need to go. By far the most common protocol used here is "Internet Protocol" (IP) which comes in two flavours that are still in use, namely IPv4 (which adds *at least* 20 bytes to each packet) and IPv6 (which add *at least* 40 bytes to each packet.)

- **Transport Layer.**

  This is where protocols such as TCP (Transport Control Protocol) and UDP (User Datagram Protocol) come in (since they're normally sent over IP, they're often referred to as TCP/IP and UDP/IP respectively.) In the case of TCP, this adds *at least* 20 bytes to each packet.

As you can see, any given stream of packets will contain a lot of information that has to be sent, but that doesn't actually contribute to the data received in any way. If we assume that the packet size is the typical WAN value of 1472 bytes and we're sending TCP over an ADSL link, then 15% of the bandwidth of the connection is immediately lost to these overheads, and for streams that use small packets (voice calls, for example) this figure can be much worse.

Whilst the Aritari ViBE SD-WAN system still has to be sent over the Internet, and so still has to deal with the overheads of its own packets, the way it packages up connections in a tunnel means that these overheads are massively reduced for connections passing through it. If we're talking about a single TCP file transfer across a WAN link with nothing else going on, then these factors roughly cancel out, so that the throughput with and without ViBE (ignoring the other factors previously discussed) would be very similar. However, in a typical network, where there are a mix of connections and packet sizes, the cumulative effect can be quite significant, meaning that more of the available bandwidth can be used, more of the time when using Aritari's system.

ARITARI
NEW WORLD NETWORKS

TEST RESULTS

Whilst theory is great, there's nothing quite like seeing the results of all of this based on real-life testing. Unfortunately, every network is different, and every use case is different, so it's not really possible to give an answer to a question such as "what is the typical improvement I can expect." Here, then, we present data produced by sending files across a simulated 100 Mbit/s symmetric network, with a round trip latency of 100mS (which is a typical value that might be experienced when going from a last-mile connection in the United Kingdom to a server based in New York City.) Clearly there are networks with much higher latency than this (UK to Australia is typically closer to 300mS for example) and those with much lower values (UK to UK would be more like 20-30mS depending on the last-mile technology in use.) In reality, the latency doesn't affect the results over an Aritari ViBE network at all, whereas it would affect a raw network using TCP. Just to prove this latter point, we also show a link with a 500mS latency (which you might see on a traditional satellite connection, for example.)

All of the following sections show graphs comparing the non-ViBE case on the left, with the Aritari accelerated case on the right, but with an otherwise identical test procedure (transferring the same amount of data with the same latency/loss.) In some cases, the transfer might start at slightly different times with respect to the start of the capture that produced the graphs, so whilst the X axis always shows seconds, the actual test may start at a slightly different time offset from 0. Wherever possible, the Y axis (throughput) scale is the same for both graphs, though the X axis (time) is different in order to show the same amount of data being transferred on each graph (often the accelerated test is completed significantly more quickly than the non-ViBE version.) Note, though, that although each pair of graphs show the same transfer left and right, different tests may involve different sized files in cases where the non-accelerated case would take too long to complete - the files were always either 300 or 100 megabytes in size.

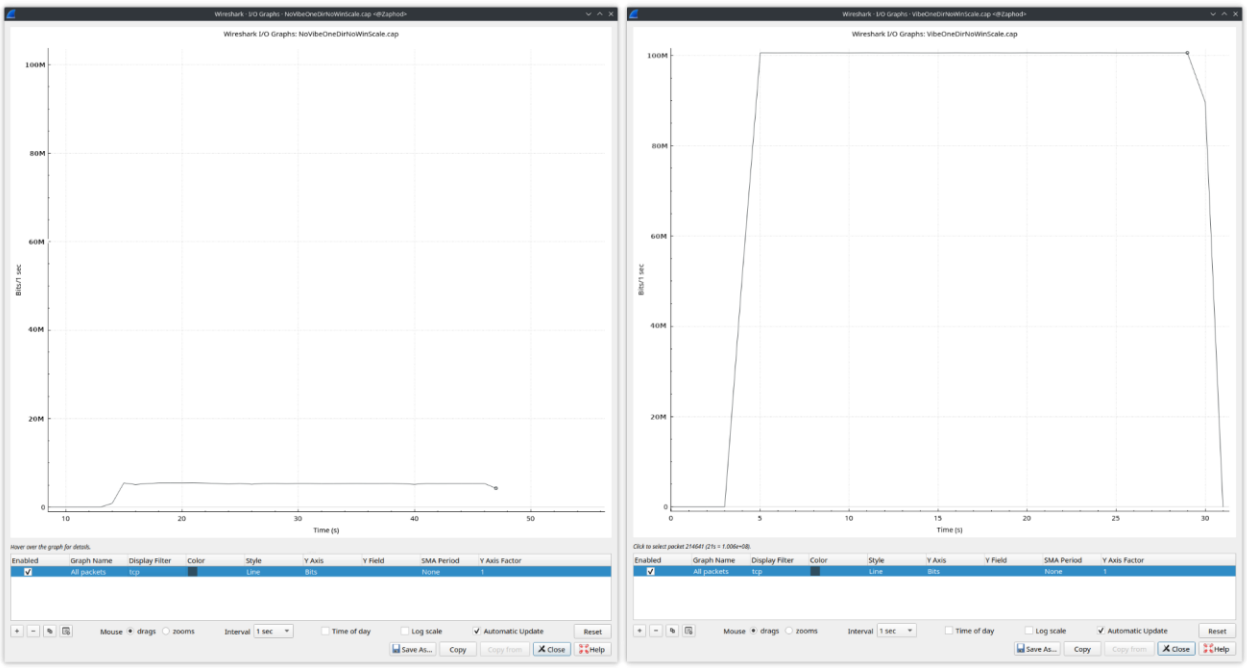
These tests were using a client and server running Ubuntu 22.04 LTS and connected using a Linux network emulator which allows latency and packet loss to be inserted into a link, but other than that no tweaks were made to any of the standard operating system settings.

Single TCP Connection with No Window Scaling

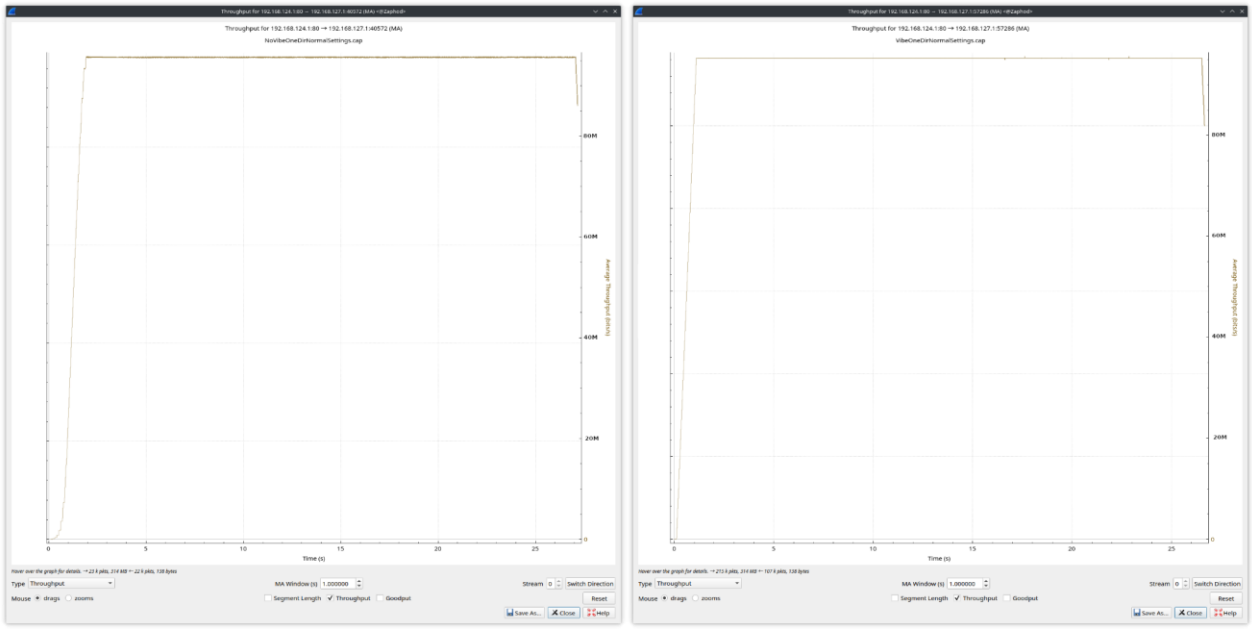
Nowadays it's unusual for any modern system to not implement the window scaling feature of TCP, though until recently there were still many operating systems that didn't implement it very well. Even so, there are still many older devices which still use 64 Kb as their maximum window size, so this is simply an illustration of what would happen if you tried to send data to such a machine.

As you can see by the graph on the left, without using Aritari ViBE, throughput is severely limited by the relatively small window size in use, whereas when a ViBE tunnel *is* in use (graph on the right) the window size in use has pretty much no effect. This makes sense

- with a window size of 64 Kbytes (actually 65536 bytes) on a 100 Mbit/s connection, it takes only 6mS to fill the window, but it will be a further 94mS before any of this data is acknowledged by the other side, essentially limiting the rate to 64K every 100mS, or just over 5 Mbit/s. When using ViBE TCP acceleration however, the sender will receive the first ACK less than 1mS after it begins to send data (because these acknowledgement packets come from the Aritari device on the LAN, and not from the remote side of the connection.)

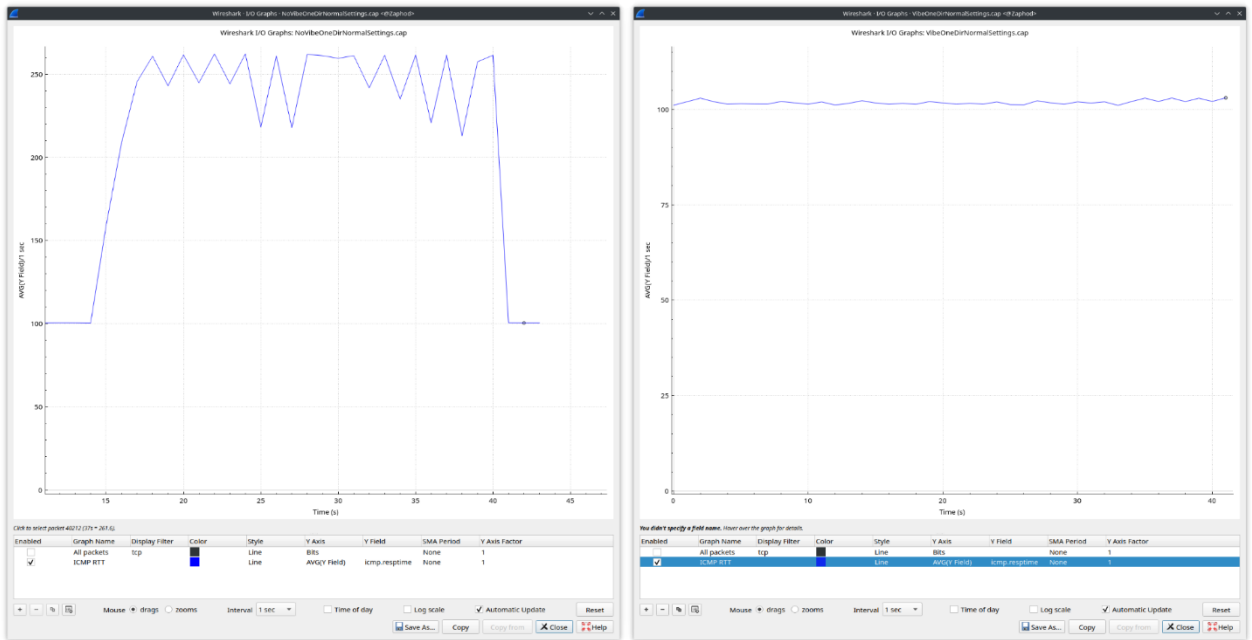


Single TCP Connection with Window Scaling



This graph shows that (at least in terms of throughput) things are much healthier, even without a ViBE tunnel in use (left graph.) With this particular client/server/link combination, the window size ramp-up period is very quick, though not instant, and the peak throughput is very close to that achieved with ViBE (right graph.) You should note though that even under these perfect conditions, there is still a small advantage in ViBE's case, completing the file transfer around half a second quicker.

Things aren't quite so rosy when you look at what happens to the network itself, however. The following graphs of latency observed using a simple ping over the link were produced at the same time as the above file transfer:

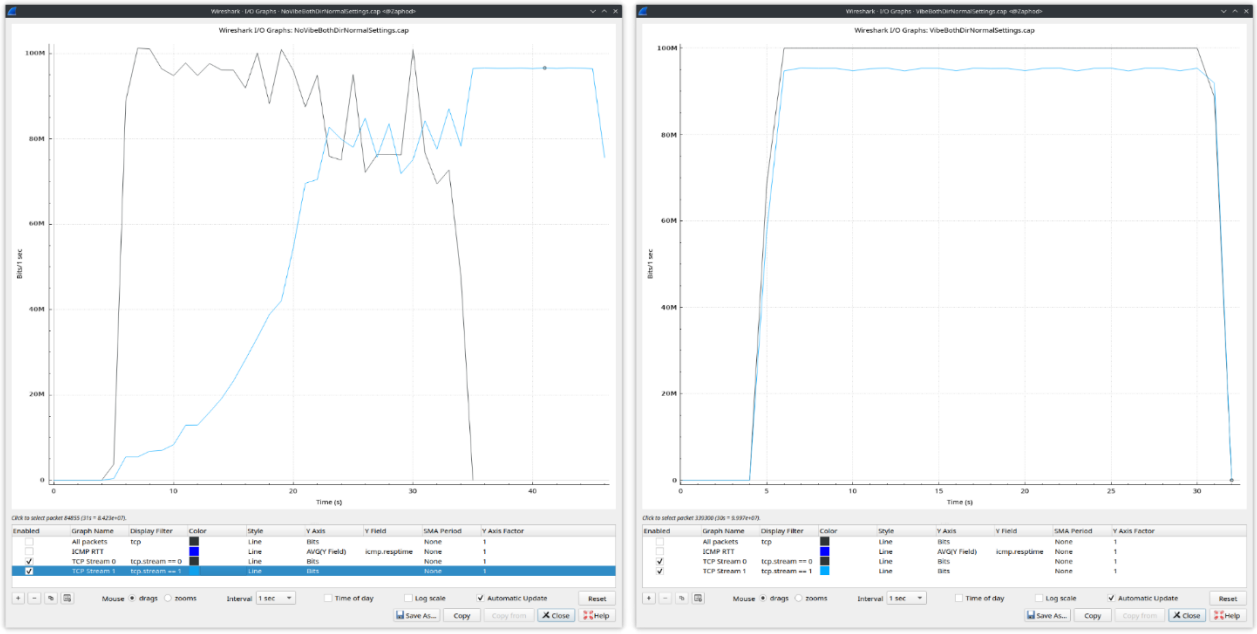


As you can see, the latency experienced by the ping test on the link jumped significantly to around 250mS when ViBE is not in use (left graph,) whilst remaining completely unaffected at just over 100mS when it is (right.)

Transfer in Both Directions Simultaneously

You may assume that data sent in one direction of a link has no effect on data going the other way, but unfortunately that isn't the case. In actual fact, if a link is being completely saturated with a single stream, the effects of packet loss due to buffer limits, or latency due to large buffers, will be felt by anything sent in the same direction as that stream, which includes the acknowledgement packets coming back from a TCP server trying to receive data in the other direction. This will cause the TCP algorithms to behave as if the connection has higher loss and/or latency than it really has, reducing throughput accordingly.

Consider the following graphs:



You can see on the left that the second stream (which is in the opposite direction from the first) is behaving as if the link is much worse than it otherwise would be (and indeed the throughput of the first stream isn't perfect, because it too is affected whenever the second stream manages to transfer at full speed, however brief these periods are.) Remember that nothing odd is going on here - we're simply connecting a client to a web server to transfer data in both directions at the same time, which will happen in a real situation all of the time (a file being uploaded to the cloud at the same time a one being downloaded, for example.) The second stream is affected much more than the first, simply by virtue of the fact that it started later, and hence was already seeing the effects of the first using the link... so the first stream completes in around 30 seconds vs the second which takes just over 41 seconds (notice that the first stream starts around 5 seconds after the start of the graph.)

As you can see from the graph on the right, when accelerated by Aritari, the streams have no such problems, both completing in around 28 seconds. Note that although from the second graph it appears that the second stream is transferring at a slightly lower rate than the first - in actual fact this is just an anomaly produced by the way the capture works. These captures show not only data, but overheads too, which are seen slightly differently when sent vs received... for confirmation you can see that this also affects the left graph (when the first stream finishes, the second shows the same slightly lower throughput) and also that the streams on the right graph finish at the same time.

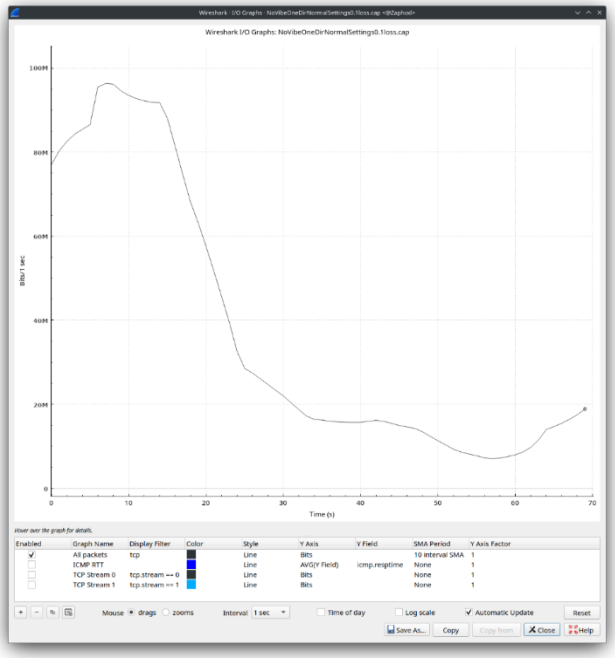
Given that real networks will be sending and receiving data all of the time, you can see how the Aritari ViBE SD-WAN can improve network responsiveness and throughput even if that network seems to be otherwise perfect.

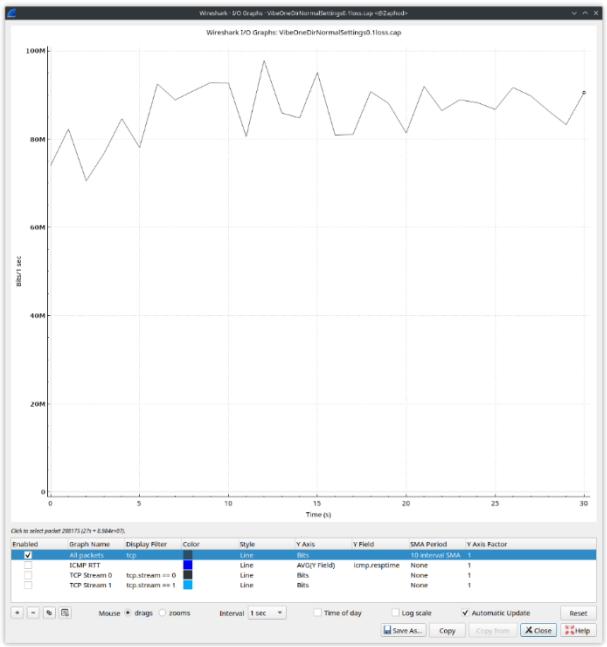
Packet Loss Tests

Let's now add some packet loss into the mix, to see how that affects things. In the real world, packet loss can occur for all sorts of reasons, including as we've seen simply because routers are managing traffic flows. However, at certain times this loss can be quite significant - maybe there's a short term increase in utilisation of a link somewhere, or perhaps you're using a radio or cellular link which are particularly prone to these issues.

We consider two different levels of loss - 0.1%, which is quite a low level than can easily be experienced on a regular basis, and 1%, which is more likely to happen when there are particular issues somewhere along the path, though this can include momentary congestion so again it isn't as uncommon as you'd hope. In both cases, the loss is applied in both directions equally, so the 0.1% case is actually 0.05% in each direction.

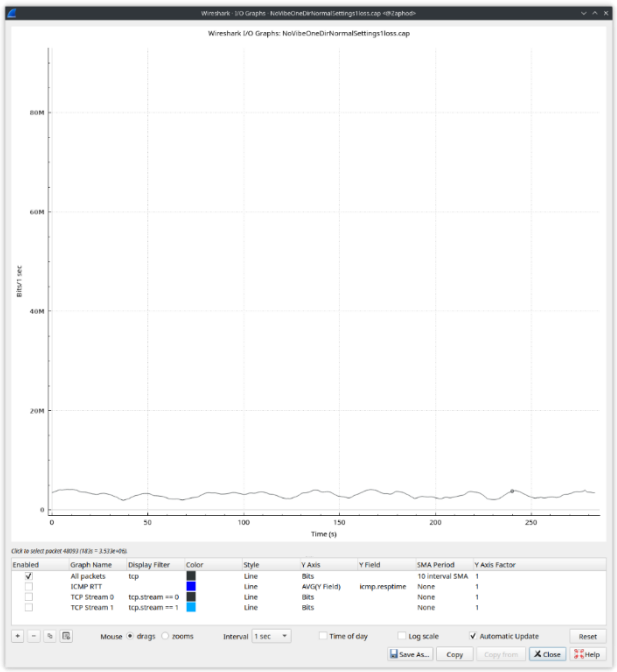
0.1% Packet Loss

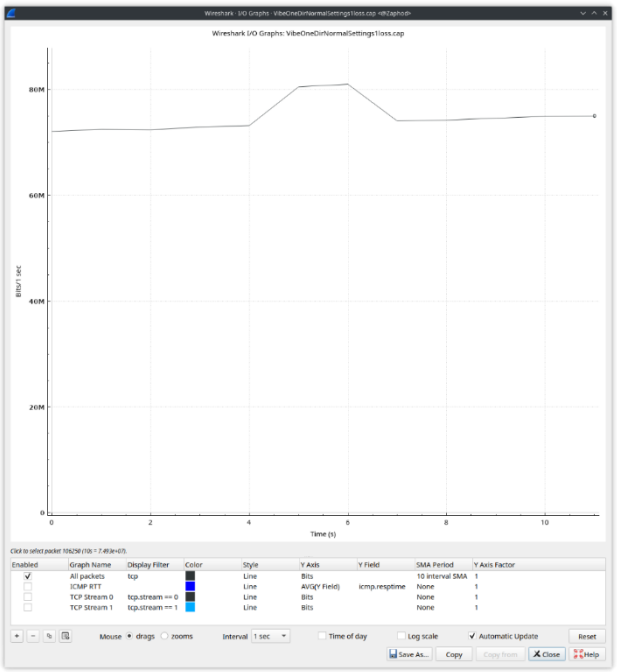




Even at this relatively low loss level, you can already see that the non-accelerated network is struggling. Things start off well, but then at some point in the stream, a packet or two go missing, triggering TCPs congestion control response, and as a result the throughput is significantly impacted. Again, the Aritari accelerated case shows no such issues, completing the transfer in less than half the time.

1% Packet Loss





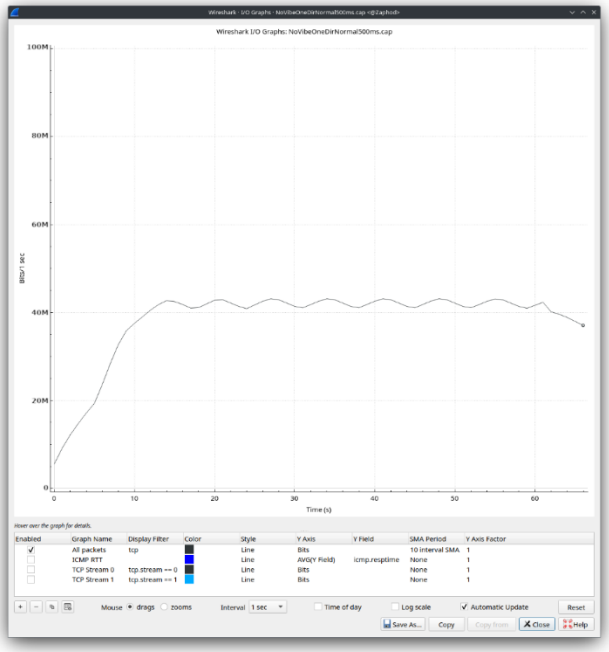
Here you can see a huge difference, with the non-accelerated transfer taking around 25 times longer to transfer this 100 megabyte file. As has been mentioned, a loss figure of 1% is not that uncommon, especially when you consider that such an event may only happen for a few seconds at a time, briefly killing any transfers in progress when it does and causing a recovery period when it returns to normal.

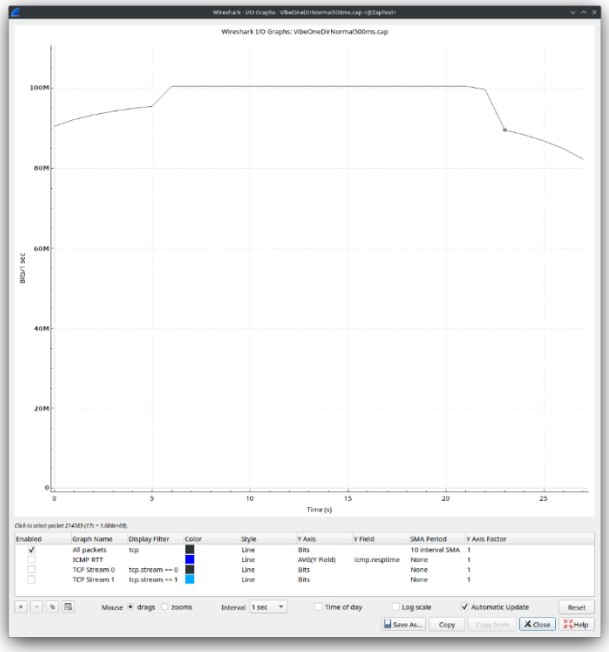
Geostationary Satellite Link - 500mS Latency

Finally for the testing, let's consider a traditional satellite link. These use satellites in geostationary orbit, whereby they sit in the same place above the earth at all times.

Unfortunately, the distance involved is quite large, so the delay in getting signals up into space and back again will very often be greater than half a second. As has been shown already though, such a high latency can also be caused by router buffer management, especially if the underlying latency is high in the first place. Earth based communication between, say, London and Sydney, can already exceed 250mS.

No Packet Loss





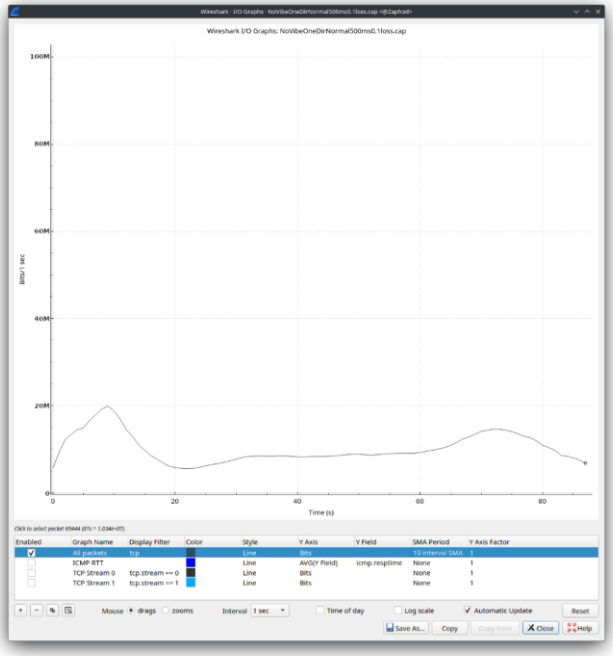
Here we have the ideal case of just a single transfer in one direction. You can clearly see two main aspects of TCP operation:

- The "slow start" mechanism, whereby the window size is slowly increased whilst no packet loss is experienced.
- The maximum window size limits the throughput to a particular rate, in this case around 40 megabits per second. Increasing the latency will reduce this limit proportionally.

It's worth reiterating that different operating systems, and indeed different versions of those operating systems, will have different limits on the window size (which can also sometimes be tweaked.) However, this test is using Ubuntu Linux 22.04 LTS with "out of the box" settings, and generally speaking Linux is very good with respect to TCP implementation when compared to other systems. You can imagine what would happen here if you tried to use a system without window scaling at all!

0.1% Packet Loss

Any packet loss on such a high latency link, without the aid of acceleration, will have a devastating effect on the usability of it. For this reason, we're only showing the 0.1% loss case, which already has a stark contrast between a standard network, and one accelerated using Aritari ViBE, with the latter completing the transfer nearly 7 times quicker.



SUMMARY

We hope that this has been an interesting insight into the workings of TCP and the benefits that using an Aritari ViBE SD-WAN solution can bring. Clearly, the results will vary depending on your specific use case and the quality of the underlying network, but even with a seemingly perfect network there will be improvements, and at times very large ones.

However, most networks are not perfect, certainly not all of the time, so above all ViBE provides consistently good results when even the best connections do not.

Aritari ViBE can be deployed as a simple VPN for a home connection, or a complete SD-WAN solution for a global business and can be installed within a customer's premises for both data centre and satellite sites, or as a managed or white label service on physical hardware or a virtual cloud infrastructure. Whatever your network topology using whatever technology, ViBE can be tailored for your use case.