Knowing You're Network Ready

7 THINGS YOU NEED TO THINK ABOUT WHEN CHOOSING A WAN EMULATOR

For years, people talked about "WAN Emulators" because basically you had your own network: the LAN (Local Area Network), and everything else was connected using the WAN (Wide Area Network). The WAN Emulator was/is a device or software that lets you actually run your application in the LAN "thinking" it's running in a WAN.

The reason for its existence is that basically, for applications, LANs were fast and great, and WANs were slow and laggy; Applications that worked great in the LAN could be awful in the WAN.

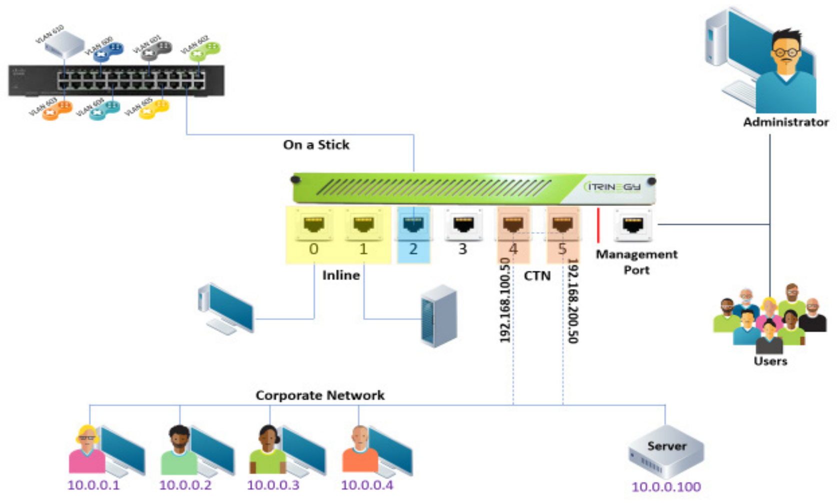
That all remains truer than ever today, except there are lots of network types now (2G, 3G, 4G, 5G, Satellite, Cloud, MPLS, Private Circuit, Internet, LoRaWAN) that we don't really think of as WANs. So the term WAN Emulator has morphed into the more general "Network Emulator", a device or software that lets you run your application in the LAN "thinking" it's running in the network type, or types (if you string segments together) of your choice.

And it's still absolutely true that applications that run great in the LAN can be awful in Non-LAN networks – for example a 4G Network.

And so the point of the Network Emulator is to allow you to find out (test – if you use that term) if your application is Network Ready for the actual network it will be used in, even with that network in its worst state; Hugely time and cost saving compared to any attempt to test in the real network.

This brings us nicely to 7 Things to think about when choosing a Network Emulator – and I won't be looking at things like how many, or which, "impairments" it has which will be the subject of another blog.

1: How are you going to insert the emulator into your network?
An odd thing to look at first, I hear you say, but actually really important. You probably already have your application running in a live network, or test or development environment, depending on the type of application and organization you are. In a test lab you can pull cables, and potentially do anything you want, but as we address applications that already live in networks, including DevOps we often can't (we're not allowed to) pull any cables. Instead we'll be needing our Network Emulator to be able to "fit in" and bridge, route, do NAT (static/dynamic), run "on a stick" and even combinations of those network types, so that we can easily insert it into our network with minimal/no disruption to the existing environment.



2: Ease of use
There are quite a few products that I use, both frequently and infrequently, that are frankly hard to use and this makes them really painful – I won't name names. It's especially true for products that I don't use constantly (as opposed to email) as there's always a getting up to speed issue.

So what can really help here with a network emulator is lots of pre-prepared network types e.g. 2G, 3G, 4G…, City-to-City like Chicago to New York etc., that can be quickly selected to get you going.

If we're trying to create a more sophisticated network structure, like a multi-link, mesh, hub & spoke, that needs to be fast and easy to do too with Out-of-the-Box Configurations and Wizards to help us.

When we want to test our application in certain predefined scenarios which vary the network e.g. a good 4G changes to a lossy 4G (worse signal), then the product should allow you to quickly create these scenarios using easy to use automation, without immediately requiring scripting, so that you have seamlessly transition between the two different network experiences.

3: Realism / Real World Impairments
I know I said I wouldn't talk about impairments (an individual effect on data transmission e.g. gaussian latency), but as I talk to some users the number of impairments offered becomes a deciding factor e.g. Product X has 200 and product Y has 300, so Y must be better than X, right? No!

This is just like the argument that developed with digital cameras – a 12 megapixel camera must be better than an 8 megapixel one, right? Also No!   Just a little reading says that the quality of the lens matters (what's the point of having 12 million blurred pixels?). And also for the same size internal screen (CCD) less light falls on an individual pixel so low light performance might be weaker with more pixels.

We're not here to talk about cameras, but the point is still there: totalling impairments is pointless – what matters is that you have the ones you need, and that the ones supplied are valuable to Real World network conditions.

4: Rapid Analysis
How do you know how much your application has been affected by the particular network conditions you "dial in"? Well, some things are obvious – if it crashes or times out in these network conditions you'll have an immediate indication and both manual and automated tests will fail.

If it runs slower, then the question is how much slower? Acceptable or not acceptable?

Here, having some strong reporting with your network emulator which gives you response times, for example, will stop you from being tied to the stopwatch during testing.

A predictive analysis report may save you even more time by suggesting what will happen when you, for example, move your application server into the cloud from being on premises.

5: Integration
If you're running or testing your application manually i.e. with real users, for example in UAT (User Acceptance Testing) this may not be a priority.

However, for most testing, people are a very expensive resource, especially if you do have an automated testing approach, DevOps methodology and/or do Continuous Integration / Continuous Test as we do at iTrinegy. In these cases, it's vital that your product is easy to integrate. Two things are vital here:

- Insertion into the network – which we covered in point 1
- Being easy to control your network emulator from other products

We'll discuss the latter as Network Insertion has already been covered above.

The kind of products you'll want to integrate with are:

- Test Automation products – test application functionality, like Selenium
- Test Load Generators – test application performance when loaded
- Your own scripts – often written in Python
- Shell scripts
- Continuous integration software – like Jenkins

Depending on your automation needs, this requires a good selection of methods to control your Network Emulator:

- REST API (sometimes referred to as RESTful API)
  - use http(s) methods GET, PUT etc to control the product
- A command line interface (CLI)
- IP Sockets Interface

Which is most valuable depends on the tools you're using, but increasingly the REST API is the preferred method.

Just before we leave this topic – whichever API you choose be careful of what I call "checkbox" functionality. "Do you have an API?" – answer yes, but what they "forgot" to mention is that you can use it to start and stop the product, but not much else. Make sure it's got all the features you need for your automation.

6: Single User or Multiple Teams
Are you happy that only a single user can use your network emulator at a time?

Wait, I'm not talking about sequential use, where when the first user finishes another can start. The problem with this is contention for the Network Emulator, and therefore a greatly increased test cycle length.

If you have an emulator that's usable by multiple users (or teams) simultaneously, then you can really cut down on those test times, or save money by not buying multiple emulators.

There are some implications in what to look for in a Network Emulator here:

- Users should be able to start, stop and update their emulation without affecting the other users – just as though you have multiple Network emulators
- You'll need a way of splitting up your Emulators physical ports (or vNICs) into Subinterfaces, Virtual Ports or Soft Ports (depending on the terminology used)
- Your Emulator product will need a security system to stop users "treading on each others toes"

7: Support
Last, but absolutely not least.

We all love Open Source (and freeware), myself included. I use Linux loads!

But, when it comes to support, you're on your own. Lots of googling (other search engines are also available) for issues. That might work for Linux, but for lesser used products, like network emulators, there's less information out there and chances are you won't find an answer. You could then post onto forums and hope someone gets back to you in a timely manner.

A second issue is who's looking after the security of the product? That's maybe fine if it's going into an isolated test lab but for Devops, Continuous Test, Development – not fine! And we get more and more questions about security even being deployed into test labs – it's rightly a topic of high importance.

And thirdly, is it being updated with the latest network types that you need – does it have 5G support for example? You can always post the request for this onto a forum.

So, having a properly supported product, from a well established vendor, should address all of these issues with a development path and regular patches (including security updates, bug fixes and new features), but there's even more:

- You should be able to get "how do I do this?" help by phone, email etc
- The vendor will have calibrated their product on the hardware they provide so that you know, for example, that when you ask for 20ms delay, that's actually what you get! [With network emulators being quite demanding on resources throwing them onto old hardware won't cut it – and you'll need to calibrate to boot]
- Where the product is supplied as a VM, OVA, software or similar that vendor will be able to provide you with minimum and typical resource requirements
- And then there are quality standards – look for vendors who care about quality – ISO 9001 anyone?

So, choose your vendor carefully – your relationship with them will be a long one – ensure it's a happy one too!

Ref: 7ThingsBlog-18052021