OpenNaaS and Virtual CPE:
a new way to connect to the Internet

Ole Frendved Hansen
DeIC, DTU, Building 356, DK-2800 Kgs. Lyngby, Denmark.
e-mail: ole.frendved.hansen@deic.dk
Peter Lavin
Computer Architecture & Grid Research Group, Lloyd Institute, Trinity College Dublin, Ireland.
e-mail: lavinp@cs.tcd.ie
Pau Minoves
Fundació i2CAT, C Gran Capità 2-4 Edifici Nexus 1, 2a planta, desp. 203, 08034 Barcelona, Spain.
e-mail: pau.minoves@i2cat.net
Bo Peng
Department of Electrical and Electronic Engineering, Merchant Venturers Building, University of Bristol, Woodland Road, Bristol, BS8 1UB, UK.
e-mail: bo.peng@bristol.ac.uk
Dave Wilson
HEAnet, 5 George's Dock, IFSC, Dublin 1, Ireland.
e-mail: dave.wilson@heanet.ie

Paper type

Technical paper

Abstract

While we've successfully virtualised and automated large parts of NREN networking, it's often difficult to realise the full benefits when we still have to build dedicated layer 3 infrastructures out of physical building blocks like routers. This is shown in the mountain of physical equipment that is needed to manage the boundary between an NREN and its clients. By developing the OpenNaaS software, we try to solve this problem in a generalised way, using already deployed technologies where they are available. OpenNaaS is a framework from which the appropriate parts can be selected, interfaced with existing services, and deployed alongside existing services. Network operators will find here a way to rapidly deploy new services that require their own topologies.

Keywords

NaaS, OpenNaaS, network, CPE, university

1. Introduction

1.1. Background

Over the last decade or so, NRENs have moved from being predominantly fixed IP networks to suppliers of dynamic services on a flexible backbone, of which the IP transit network is "merely" one user. This has been accomplished by the deployment of virtualised and automated solutions on different layers, which are then turned into products. The products are accessible to users, sometimes through very simple interfaces (e.g. one RJ45 port per service) and sometimes through more complex and sophisticated demarcs (e.g. VLANs provisioned based on a request processed from an API that respects multidomain boundaries.)

It's a fine aim. Abstract out the network sufficiently that everything we need to do on it becomes just another service on this generic framework. As versatile as packet switching is, it's a boon to be able to overlay multiple private networks on a single infrastructure using dynamic layer 2 circuits, or other virtualisation technologies. But we have run into a limit.

1.2. What are the problems?

IP technology is so ubiquitous now that even when our customers make private networks, the likelihood is that they'll be implemented using IP, typically commodity hardware that is configured to be appropriate to the task at hand. To make an IP network scale, though, you need routers. Theory says that our users can create dynamic networks in minutes out of mere photons, but practice seems to result in a to-do list that begins "First order some routers, and wait for them to be delivered." This, obviously, is a drag on the effective use of dynamic services.

There is another drawback, though. Where we do try to implement services that span several layers, we end up with a large and increasing pile of equipment that is necessary in order to drive the services. At a typical NREN's customer site, one may find:

- Transmission equipment providing the optical connectivity for the NREN
- Two IP routers implementing a resilient access boundary for the NREN
- Two more IP routers implementing the resilient boundary of the customer's internal network
- Two switches providing non-IP and dynamic services from the NREN
- Two firewalls which implement the customer's security policy

All this to provide reliable connectivity to one customer. Is there anything we can do to rationalise this? Every part of this set up came about for good reasons:

- The transmission equipment is specified by nothing less tractable than the physical characteristics of the fibre in the ground - a certain level of power is needed to light it, and that's that.
- The firewall is not the only way to implement a security policy, but for a university network trying to provide enterprise-level reliability to its users, it's very often the most cost effective way to do so

Other parts of this setup are the product of hard-won lessons.

- We double the access routers, campus boundary routers and switches, because we have seen the consequences of a single point of failure, and we're willing to pay to avoid them.
- We separate the NREN's access from the customer's campus network not for physical reasons - more routers does not mean more bandwidth - but because provider and customer each need to manage their own networks without impacting or depending on the other.

Let's look more closely at this last item. The NREN and the customer both use separate routers on their network not for physical separation, but in order to manage them separately. Each needs to use it own routing protocols, monitoring systems and authentication policies in order to ensure that the service is functioning correctly. The campus OSPF deployment must not be allowed to impact, or be impacted by, OSPF running on the NREN's backbone.

But these are all software functions. Software can be virtualised.

2. Tools at our disposal

Our experience deploying services in the past suggests that there are three ingredients to the "secret sauce" we need to make this work.

The first is virtualisation. Anywhere we want to take a single physical resource and present it as separate resources to separate users, we use virtualisation technology. In practice we see this working today not just in server systems such as VMware, and in our layer 2 networks where separate circuits are configured for different users, but also in layer 3 where a single JUNOS router can be configured with multiple logical systems.

The second is automation. Virtualisation, unfortunately, can get out of hand. While it is possible to create many different virtual resources, one still has the problem of managing each one of them, and this is a problem which does not scale when performed by hand. If you automate your provisioning, though, then multiple advantages appear.

The configurations are likely to be broadly similar, and certainly very well understood. This has a remarkably deep impact on change control. With a human operator, one must always be prepared for some level of error; the very best among us have accidentally disabled connectivity to the device we were configuring. If you have a provisioning system operating on your production network, it is possible to have a very high degree of confidence in the changes it is making because it has, perhaps literally, made those same changes a thousand times before.

Another advantage of automation is the sheer speed at which services can be provisioned. Even where speed of provisioning is not a direct customer requirement, this does directly affect the amount of time operations personnel spend in repetitive tasks. A service that is configured by means of a pair of drop-down forms is likely to take considerably less effort to configure than one which requires configuration to be hand-built based on entries in a spreadsheet.

And one more significant advantage of automation is that one can expose the service by means of an Application Program Interface (API). This can be stated simply: if one is developing a service that one ever wishes to integrate with other products or services, then it doesn't necessarily need an API on day one - but it must be automatable.

There is one more ingredient to our secret sauce though, and that is incremental deployment. Regardless of the advantages that we can see once our idea is fully deployed, the journey that it takes to get it there can be made markedly easier provided that it does not disrupt existing workflows along the way. If it cannot be laid alongside the existing, used services, at least for long enough for it to become a proven product that can subsume the old services, then the path to deployment becomes an uphill struggle.

As well as these three ingredients, we have some concrete tools at our disposal.

You can't have a network without connectivity. Fortunately, there is already a Bandwidth on Demand [AutoBAHN] system which allows us to create circuits, not just between NRENs, but inside some of them. In Ireland, for example, this is deployed on HEAnet's production layer 2 network, so we can use this system to automatically provision circuits between two given points in the HEAnet network, as well as between a pair of NRENs if we so choose. Not only that, but the system has its own API, so we have the opportunity to make these requests from inside another tool.

Finally, JUNOS routers support logical systems, which allows one to configure a virtual instance of a router with its own interfaces and its own instances of its routing protocols. Neither is this the only way to accomplish this - virtual machines running open-source routing software are an obvious target.

2.1. Making a framework

So now that we know the tools available to us, the question becomes: how can we bring these together into a product that we can use to solve the problem?

The OpenNaaS software is a framework which layers the technologies on each other to make a solution appropriate to the task at hand. The physical resources themselves are managed directly by the Resource Manager Platform. The Resource Layer exposes various abstract resources such as Router, Network or BoD Network - so a network is not composed of individual physical routers and circuits, but resources which perform those functions, and are likely to be implemented by virtualised instances of themselves.

Finally we have the Remoting layer, which is the external interface to the system. In particular, the OpenNaaS software provides both a command line interface (CLI) and an API. The model we have chosen for our use cases, as seen below, and the model that we expect that most users will choose, is to build a GUI that is specialised to the task at hand, which drives the more generalised platform underneath. This has a number of advantages; not only does it simplify the task of developing a system down to extracting the necessary functions from the framework, but where functionality is at first missing from the GUI, it can still be reached by means of the CLI.

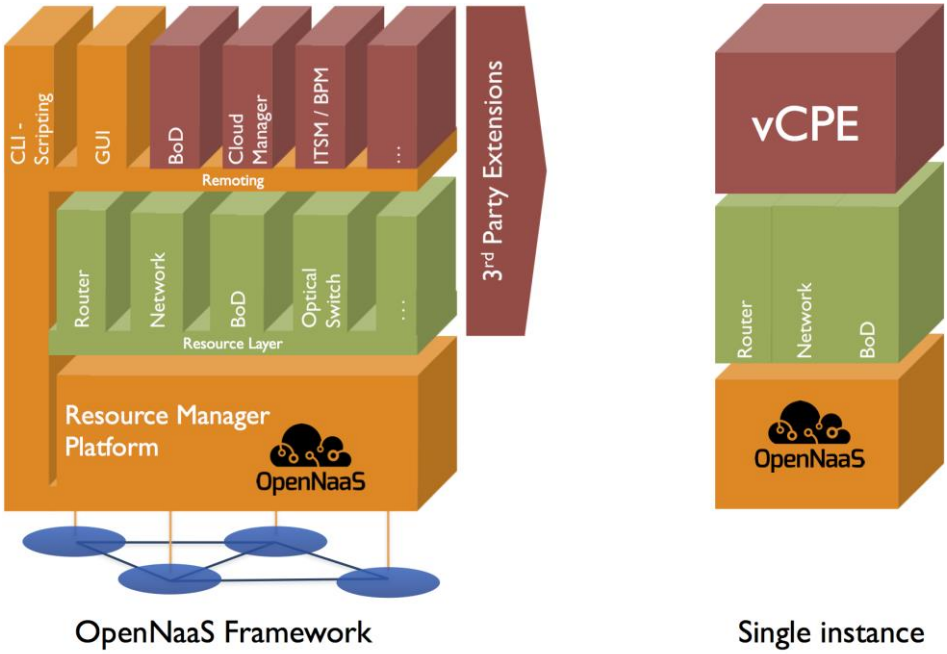


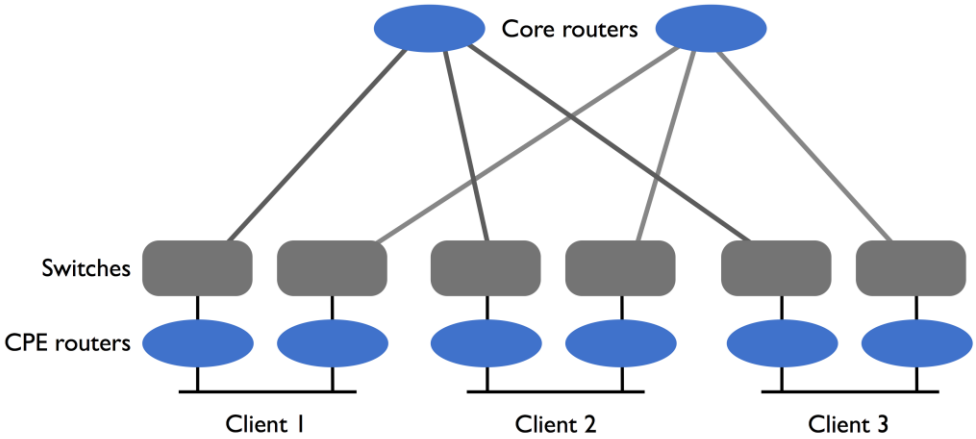
Figure 1: OpenNaas Framework and one example instance

3. Ways to solve the problem

The MANTYCHORE project has focused on solving five specific use cases in the development of the OpenNaaS software. In this paper we will use two of them in order to illuminate the solutions. In line with the previous section, we have made a specialised GUI to perform these two cases, which drives the general OpenNaaS software to implement them.

3.1. Several customers with one provider

One scenario where we can try to reduce the amount of physical equipment deployed is where a single ISP (say, an NREN) connects several different customers with similar configurations. A typical layout is shown in figure 2, where

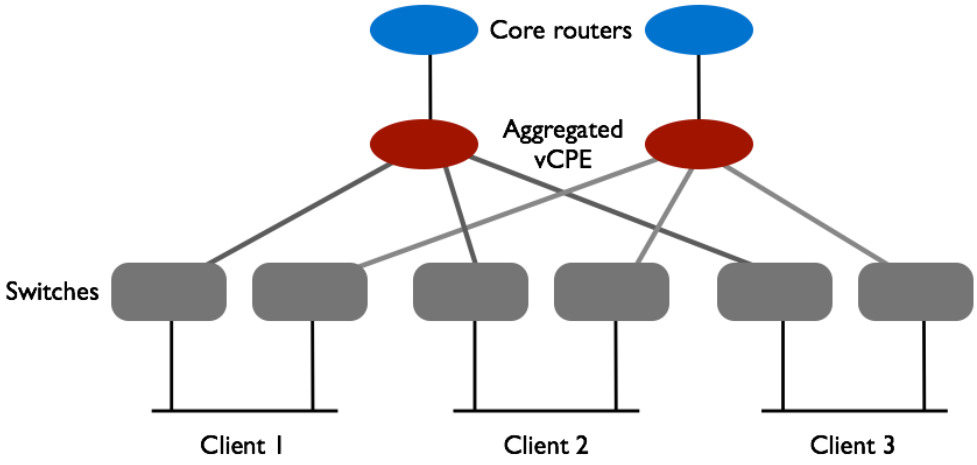


the NREN deploys both CPE routers and switches on site, in order to provide both an IP boundary and non-IP services.

Figure 2: Abstracted double-star NREN layout

Physically, we remove the CPE routers and replace them with aggregated routers close to the core, which perform the same functions for all connected clients, as in Figure 3. Of course, as discussed above, it's not adequate for a single

router instance to try to perform this task, as the router will in effect be a part of the various clients' internal campus networks, Therefore, each client is assigned a pair of logical routers which are hosted by the physical router, but are in



effect a part of their own network, isolated from the rest of the logical routers in that physical device.

Figure 3: Abstracted NREN layout with virtual CPE

The twin characteristics of this setup are: the virtual CPEs must be consistent enough with each other that they can be managed by a single operator; but each virtual CPE must be adaptable enough to integrate with the client's own network, including whatever routing protocols they use internally.

We accomplish this by providing a single GUI with two views. When a network operator logs into the system, they are presented with a view of all the vCPEs that have been deployed, and the ability to create new ones. Each follows a template that is consistent with the use case - again, of course, while this is quite strict in the GUI, the GUI can be changed to allow any operation that the software supports, and the CLI can be used for one-off changes.

A system such as this must of course sit alongside other technologies, and leverage them where possible. We have the advantage in this case of not having to implement our own bandwidth provisioning system, because one already exists. HEAnet's point-to-point provisioning system is already integrated with the GÉANT Bandwidth on Demand system (GÉANT Dynamic) so we may use the BoD API in order to request circuits, even though in this case the circuits lie entirely within HEAnet's domain.

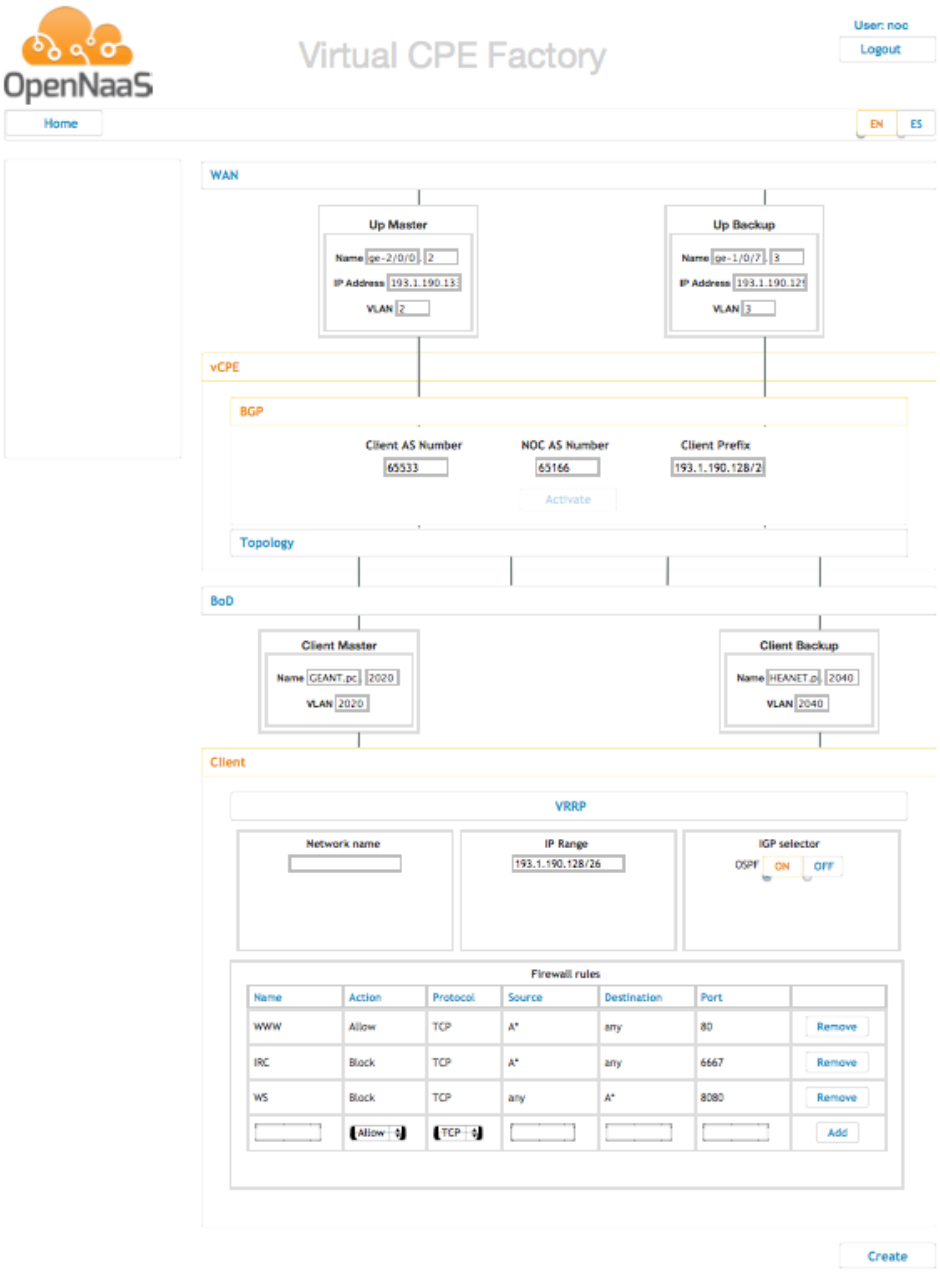


Figure 4: Routing configuration in the virtual CPE GUI.

We also take the approach that duplication of information should be avoided. Once an operator provides a piece of information relevant to the setup, it should not be necessary to re-state that information elsewhere if it can be inferred from what was already provided. This can be seen in the screenshot in Figure 4 which shows the parts of the GUI relevant to routing. By the time the client-provider BGP information, address range and interfaces have been defined, all that remains to configure OSPF is the question: on or off?

This provides the necessary flexibility to the operator. But what about the client, who must integrate this logical router into their network? For this, the OpenNaaS tool provides a separate view into the same infrastructure.

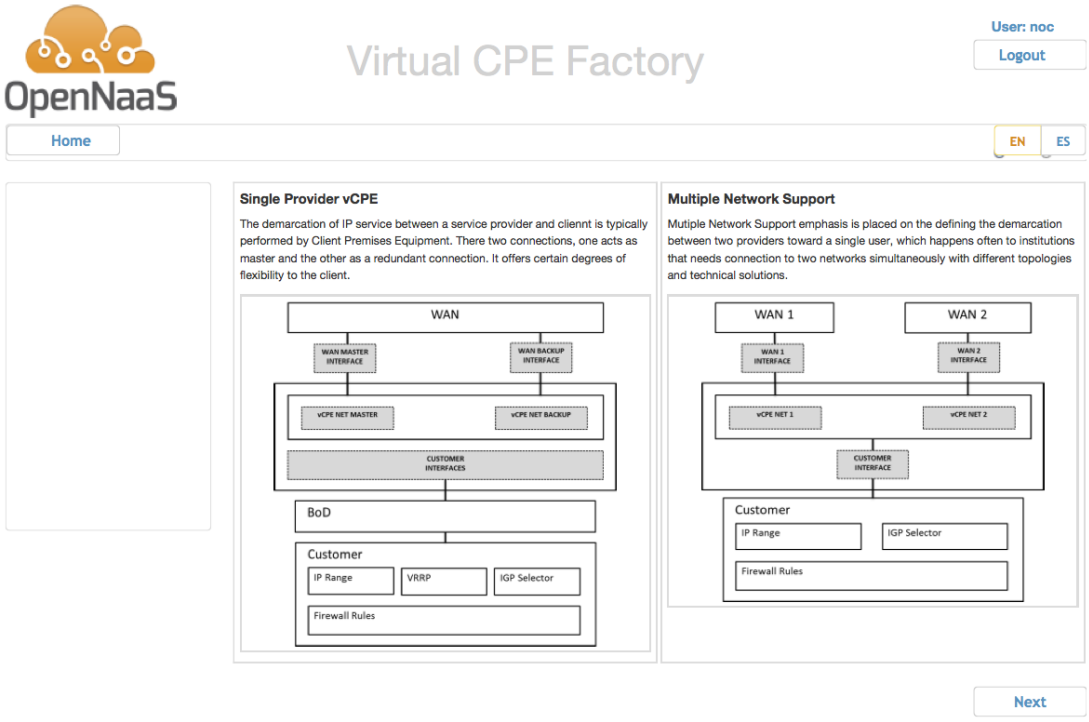
There are portions of a CPE's configuration that must be set by the network operator, or at least in agreement with them. The link addresses, announced networks, VLAN tags on any virtual circuits, are fixed when the connection is commissioned and, if any changes are made by the end user, they will not work unless they are also reflected by their upstream provider. But there are other parts of the configuration which are clearly a part of the client's internal network - e.g. the gateway addresses, routed subnets and VRRP addresses assigned to the downstream LAN interfaces, and what

routing protocol is used on the network as a whole. These must be set by the client, and the client must be able to change them. It is wasteful if this mechanism for change begins with a phone call to the upstream ISP.

The OpenNaaS software supports the principle of multiple levels of access, and this is reflected in the GUI. (Federated authentication of a web-based front end to a RESTful service is a thorny problem, but one we are working on within the project.) The client is able to log in to the same system that the network operator uses, and from their they can view and edit the resources that are assigned to them, without impacting either their upstream connectivity or other clients using the same physical equipment.

3.2. Single customer with multiple providers

A closely related use case is that being investigated by DeIC, the Danish e-Infrastructure Cooperation, where a single



institution must connect to two independently managed networks. Currently this must be delivered as multiple connections either with no CPE, or with separate CPE.

Figure 5: Choice of models

However, the underlying problem to be solved here is very similar to that discussed above; a single piece of equipment that must perform duties for, and be managed by, more than one operator. So while we expect that each use of OpenNaaS will have a specialised GUI, in cases such as this it is possible to leverage work already done on one use case to satisfy another, with incremental changes to the interface and to the underlying template that implements it in OpenNaaS.

Figure 5 shows the choice provided to the user when they log in to the software as it currently stands - either model of network can be created and managed by the same software.

4. Conclusion: The future

What we've shown here is the automation and virtualisation of one common operational action in a NOC - the connection of a customer using CPE. It's an improvement on a number of metrics. The process of creating the virtual CPE and setting up the underlying connections takes about 4m30s, determined mainly by the speed of the underlying bandwidth provisioning system; this is as opposed to lead time for ordering and deploying physical equipment which (depending on local stocks) could be hours or weeks.

Work remains to be done in operational pilot projects. It's clear that there is opportunity for cost savings in capital equipment, for example, and estimates can be made, but this needs to be tested in a fully operational environment. Also, there are likely to be some operational adjustments that result from moving the BGP session away from the physical demarcation point at the customer's site. These are surmountable obstacles, but the best process to handle them will be determined in an operational pilot.

So virtual CPE is an excellent starting point to explore NaaS in an operational environment. There are clear advantages to using it; it's working on a very well defined and well understood service; and it's being tested with real, production traffic with minimal risk of disruption. But these aren't the only reasons why it's an excellent first step.

By using NaaS to provide this particular crucial service, the gateway between the provider and the client networks, we also open up the opportunity for many other kinds of services that can be provided directly to the client (or, where appropriate, multiple clients as a group) such as the other use cases in the Mantychore project [Mantychore]. Virtual CPE doesn't have to merely replicate the existing CPE setup - it can go much, much further using all the same tools. All the possibilities of NaaS become available directly to the client as well as to the provider.

The most interesting applications, the services we actually deploy, are not built with bandwidth alone. Neither are they built with routers alone. This is something that applies from mundane CPE all the way up to the most exotic of dynamic services; when building a service requires an expensive purchase and a long wait in order to deploy equipment, it is hard to get the full benefit of the demand-based systems that we build. But by automating, virtualising and aggregating all layers of the network, such that all parts of the network can be spun up on demand, this changes.

As NRENs, we have been laying the building blocks for this sort of solution for over a decade. Far from being limited to virtualising on-site equipment, we now have a solution for deploying full, virtualised networks which can be managed wholly, partially or not at all by end users with authority delegated from their institutions and NRENs.

5. References

[OpenNaaS] http://www.opennaas.org/
[Mantychore] http://www.mantychore.eu/
[AutoBAHN] http://geant3.archive.geant.net/service/autobahn/about_autoBAHN/Pages/About_autoBAHN.aspx