Enhancing the BRAS through Virtualization

Thomas Dietz, Roberto Bifulco, Filipe Manco, Joao Martins, Hans-Joerg Kolbe, Felipe Huici

NEC Laboratories Europe,
name.surname@neclab.eu

***Abstract*—Broadband Remote Access Servers (BRASes) are crucial middleboxes in DSL access networks, providing the first IP point in the network for subscribers and enforcing operator policies. The number of functions provided by BRASes, combined with the key role they play in the network, means that these devices are expensive, difficult to change, and constitute a single point of failure. In order to overcome these limitations, we propose to virtualize the BRAS and to enhance it with a control interface that can be exploited by management systems in order to introduce live session migration and higher reliability. Our proof-of-concept implementation shows that our virtual software BRAS is able to handle thousands of sessions while forwarding and shaping traffic at rates of millions of packets per second on commodity hardware, and that the live session migration feature enables the implementation of high-reliability scenarios.**

I. Introduction

Network operators face the conflicting goals of providing ever increasing traffic rates while reducing the expenditure needed to run their infrastructure. In access networks, one of the key components is the Broadband Remote Access Server (BRAS), also known as a Broadband Network Gateway (BNG) [10]. BRASes carry out vital tasks such as managing subscribers' sessions, performing accounting, enforcing operator policies (e.g., traffic shaping, firewalling), and acting as the first IP point in the network.

Given their central role, BRASes represent an important source of expenditure for operators. For one, they consist of expensive, hardware-based boxes provisioned for peak traffic. Further, their unique placement in the networks means that they constitute a single point of failure: the failure of a single BRAS may cause the disconnection of thousands of subscribers, resulting in large costs in terms of technical support and other related expenses. Such scenario also means that operators are reluctant to deploy new functionality that might decrease the BRAS' reliability; even if they decided to, the fact that BRASes are proprietary, hardware-based devices means that changing their functionality is far from trivial.

A number of these shortcomings could be addressed by shifting the functionality of BRASes towards software running on commodity hardware, as exemplified by recent trends in Software-Defined Networking (SDN) [8] and Network Function Virtualization (NFV) [9]. To the best of our knowledge, however, no work has looked into providing high reliability, virtualized, software-based BRASes. To address this, in this paper we present the design, implementation and evaluation of such a virtual software BRAS. Our contributions are as follows:

- The prototypical implementation of a virtualized software BRAS built on top of the ClickOS [7] framework. The BRAS can forward packets at rates of up to 1.5 Mp/s on a single CPU core while applying shaping and ACL rules to the traffic, and can be instantiated in as little as 50 milliseconds.
- The introduction of a control interface that can be used to realize live, subscriber session migration. Our management system can migrate subscriber sessions between BRASes with zero downtime, or exploit the interface to implement remediation strategies in case of failure. For the latter, our prototype is able to restore service within 800 milliseconds without dropping any sessions.
- The implementation of suspend and resume functionality for MiniOS, the paravirtualized OS that our BRAS is based on. This allow us to *transparently* migrate thousands of sessions in a BRAS instance in about a second while handling rates of millions of packets per second, and enables interesting scenarios such as the ability to load balance sessions or perform maintenance without service interruption.

In the rest of this paper we present further background regarding BRASes and the ClickOS framework that our prototype is based on (section II). In section III we discuss the design and implementation of the BRAS, followed by an evaluation of it in section IV. Finally, section V discusses related work and section VI concludes.

II. Background

A. Broadband Access Networks

The Broadband Access Network connects an end customer or subscriber to the operator's service and core networks. An access network usually includes a last mile link, an aggregation network (e.g., based on Ethernet), and a Point-of-Presence (PoP), the place where the aggregation network terminates.

One of the most common last mile technologies is DSL [11]. When using DSL, the subscriber's home gateway establishes a session with the operator's BRAS, a box located in the operator's PoP. In a typical use case, the home gateway and the BRAS use the PPPoE/PPP protocols for session establishment. PPP creates a point-to-point tunnel between the home gateway and the BRAS, while PPPoE is an adaptation layer to run PPP on top of Ethernet. To this end, PPPoE operations are divided into (1) a discovery phase, in order to select a single BRAS on top of a point-to-multi-point

978-1-4799-7899-1/15/$31.00

technology such as Ethernet; and (2) a session phase, once a BRAS has been selected. Any packet flowing between the home gateway and the BRAS is encapsulated in PPPoE and PPP headers. The BRAS has a central role in the architecture since it implements policy enforcement (e.g., traffic shaping), interfaces with the AAA (Authentication, Authorization and Accounting) services, performs IP routing and ARP proxy functions, and, more generally, is the first point in the network where the operator can introduce new services.

B. Click and ClickOS

Our software BRAS prototype is implemented on top of the ClickOS [7] framework. ClickOS consists of tiny, Xen-based virtual machines aimed at network processing. ClickOS VMs are made up of the Click modular route software [6] running on top of MiniOS, a minimalistic OS provided with the Xen sources. Click is made up of small modules called *elements* that implement small bits of functionality, such as decreasing a TTL or counting packets. Further, each element can have read or write element *handlers* in order to read or modify element state (e.g., a packet counter element would have a read handler to retrieve the current packet count, and a write handler to reset the count).

Such elements are then connected together through *ports* into a *configuration* that can implement a network function like a firewall, a load balancer or an IP router. ClickOS executes Click configurations within its tiny VMs. Such VMs are capable of booting in a few tens of milliseconds, have a small 5MB memory footprint, and can handle rates of several million packets per second when used with an optimized Xen network back-end [7].

III. BRAS Design

In this section we describe the design of the virtual software BRAS, describing the BRAS control interface as well as its session migration mechanism.

A. Basic Functionality

The BRAS has three network interfaces: one facing the access network, one facing the core network and a final one acting as the control interface (the Click `FromDevice` elements in figure 1).

The access element sends packets to a `Classifier` that splits them among PPPoE discovery and PPPoE session packets by looking at the Ethernet protocol number. PPPoE discovery packets are sent to the `PPPoEDiscovery` element which implements the PPPoE Finite State Machine (FSM) and generates response packets that are finally sent to the `ToDevice(access)` element. The PPPoE session packets are handled by the `PPPoESession` element. This element is in charge of verifying that the PPPoE session is in place for the received packet and of stripping it of the PPPoE encapsulation.

The packet is then passed to the `PPP` element, in charge of maintaining the PPP FSM, of verifying the PPP session and of stripping the PPP encapsulation. From there, the `IPRouting` and `IP-ARPProcessing` elements handle IP communication before the packet is sent to the `ToDevice(core)` element. A packet flowing in the opposite direction, from core to access network, follows somewhat similar processing, except a check is done to ensure that there is an existing PPP session for it so that it can be encapsulated in PPP/PPPoE/Ethernet headers before delivery to the access network.

The `SessionStore` element plays a special role in this design. This element never process network packets, since its role is maintaining all the relevant state data required by the PPP and PPPoE elements. The `SessionStore` element, hence, has read/write handlers but no ports.

B. Function Modularity

Beyond this basic functionality, captured by the dotted line box labeled "basic connectivity" in figure 1, our software BRAS can be adapted to support different sets of functions. Two such functions are the ACL and Traffic Shaping groups of elements delineated by dotted lines in the figure, both of which can be effortlessly added or removed from the BRAS. The modularity inherited from Click allow us to envisage scenarios where ACL and traffic shaping functions are provided in a different device during high peak periods, and consolidated back into a single device when the workload dies down.

C. Control Interface

The implementation of the BRAS control interface is based around the concept of a session. The session state consists mainly of the subscriber's network configuration information, its policies, and the state of the PPPoE and PPP FSMs. The BRAS stores this information in the `SessionStore` element, and provides an interface to register callbacks for session establishment and session termination events.

When firing the session establishment callback, the interface provides all relevant subscriber information. Assuming that no notifications are lost, this enables an external entity to keep a complete mirror of the BRAS's subscriber state, an important feature that enables some advanced reliability scenarios.

Finally, it is worth noting that the `SessionStoreInterface` element implements a simple, UDP-based network protocol to give access to this interface.

D. Session Migration

The BRAS is the only place in the network where the subscriber session state is stored. Redirecting sessions to a different BRAS, e.g., for load balancing or reliability reasons, is a complex task that either involves custom solutions based on traffic mirroring and state replication, or requires the sessions to be terminated and re-established. To address these issues, we have implemented two mechanisms that allow us to effectively migrate BRAS sessions: transparent migration, which enables our prototype to migrate sessions without special instrumentation of the BRAS; and fine-grained migration, which requires (reasonable) changes to the BRAS but allows for more efficient migration as well as resilience to failure.

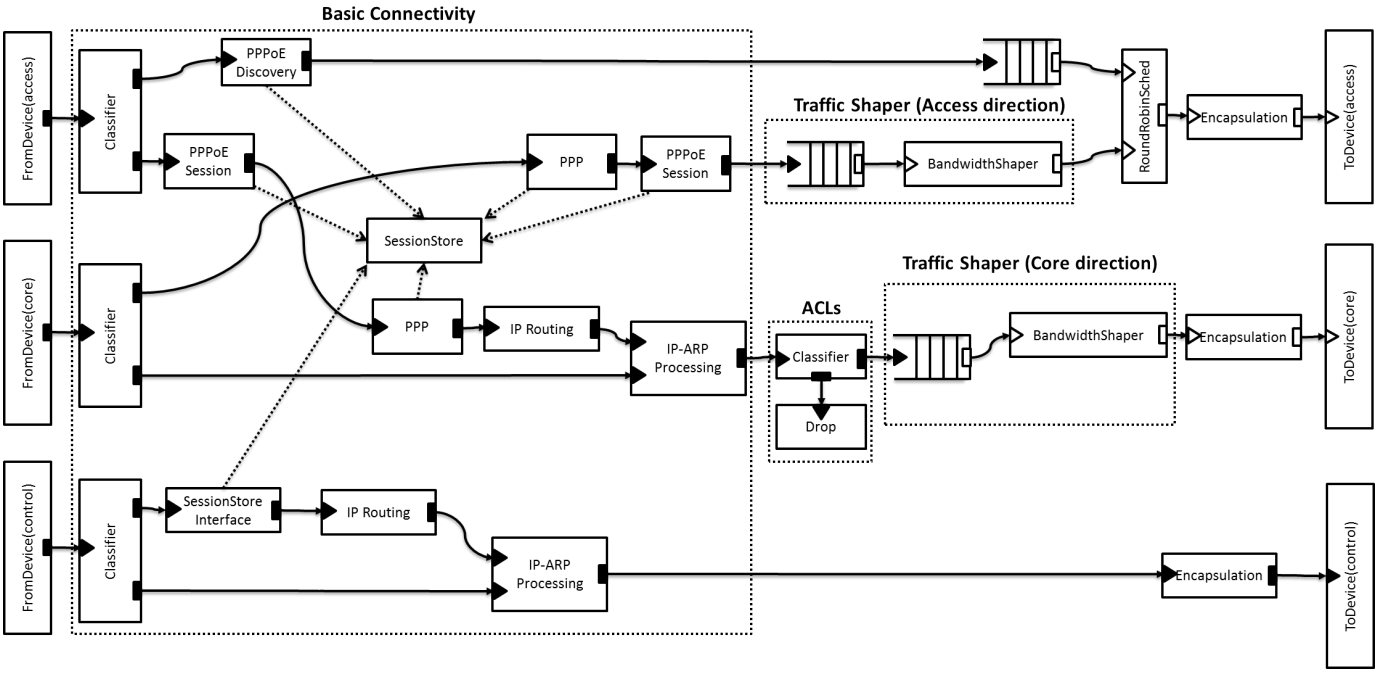

Fig. 1. Software BRAS design with three interfaces, one for the core network, one for the access network, and one for control. The design is modular, consisting of a basic session handling and IP component, a traffic shaping one and one that handles ACLs (shown in dotted line boxes).

Transparent migration: The fact that our software BRAS is virtualized opens up the possibility of using standard live migration techniques to shift session state from one server to another. In principle, live migration consists of moving the entire VM, along with all of its state, without interrupting its processing. In practice, the VM experiences a small, outage.

To enable live migration, we implement suspend and resume functionality to MiniOS, the minimalistic OS that our software BRAS VMs are based on. Further, since our BRAS prototype is based around tiny VMs, this outage period is limited compared to that exhibited by VMs based on full-fledged OSes such as Linux; we present evaluation results for this effect in section IV.

The clear advantage of this mechanism is that the VM is migrated as a black box, and so requires no special functionality from the BRAS to support session migration. This means that an orchestration and management system can move the VM and its related workload without caring about the details of the specific processing executed by the VM.

Fine-grained migration: While transparent, the previous mechanism is coarse-grained, forcing all sessions in a VM to be migrated even if only a few needed to be moved, an inefficiency that leads to longer outage periods. In addition, if the virtualized BRAS were to fail, all corresponding application state (i.e., the subscribers' session state) would be lost.

To avoid these problems, we further support fine-grained (non-transparent) session migration through the use of the control interface previously described. Beyond the ability to migrate specific sessions, this interface allows for continued operation in the face of a BRAS failure, since the subscribers' session state can be exported and mirrored in a different location. The downside is that the operator's management system would have to be extended to cope with the mechanisms exposed by the interface. However, we believe this requirement to be relatively painless and inline with recent SDN trends in which complex network functions (i.e., beyond plain switching elements) offer an interface to the network control plane [4].

IV. Evaluation

In this section we evaluate our BRAS implementation. We used a HP DL380G7 server equipped with an Intel Xeon CPU E31220 (4 cores, 3.10GHz) processor, 16GB of RDIMM PC3-10600R-9 memory and an Intel 10Gb card. A similar server is used for traffic generation and for throughput measurements. The two servers are connected via a direct cable. Throughout, we use Mp/s to mean millions of packets per second.

Boot time and memory footprint: Features like VM boot time and memory footprint are inherited from the ClickOS platform [7] which our system is built on. The BRAS boots in less than 50ms and its memory footprint is only a few MBs in size.

Throughput: To test what sort of throughput our BRAS is able to sustain, we generate packets from an external server to the one hosting the BRAS; the BRAS then forwards them back out, and we measure the resulting throughput at the external server. For this experiment we make use of two CPU cores: one for the BRAS VM, and one for the Xen networking backend.

As a baseline, we first tested the forwarding rate when no BRAS-specific functionality is in place, that is, the forwarding

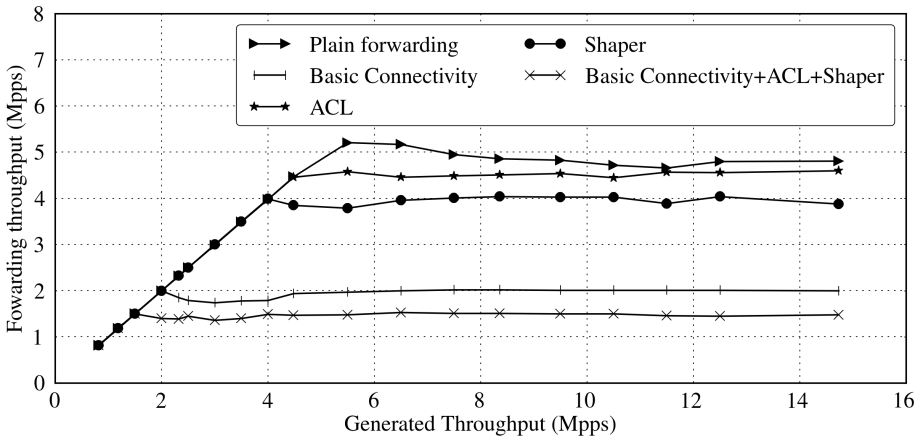

Fig. 2. BRAS throughput performance with respect to offered load. The basic connectivity, ACL and shaper components are tested individually, and then put together to derive an overall BRAS rate, resulting in rates of 2.1, 4.5, 4 and 1.5 millions of packets per second, respectively. The plain forwarding line shows a baseline rate for ClickOS without BRAS functionality. In all the cases, increasing the number of packets per second sent by the generator does not have a significant impact on the forwarding performance

rate of standard ClickOS (see "plain forwarding" line in figure 2). We then proceeded to test the three components of the BRAS (the ACL, the shaper, and the basic connectivity) individually, for which we obtained rates of 4.5 Mp/s, 4 Mp/s, and 2.1 Mp/s, respectively; these rates hold fairly steady regardless of the offered rate. Finally, putting all components together yielded a rate of about 1.5 Mp/s. In all, the virtualized software BRAS prototype is able to handle 10Gb/s line rate for all packets sizes bigger than 512B when using a single CPU core for the VM. We believe these rates to be in line with what a BRAS might experience. In addition, The rates can be further scaled out by instantiating additional BRAS VMs, something that only takes milliseconds to do).

Control interface: We evaluated the speed of the BRAS control interface and ultimately the speed of the `SessionStore` element. We configured the measuring box to run a *SessionServer*, implemented in C, which pushes the state of 14K sessions to the BRAS interface. The session installation generates 1,000 UDP packets, each of them 1400B in size. The installation of all sessions takes less than 100 milliseconds (an average of 5 microseconds per session), making it possible to shift processing and cope with failure in an almost transparent way.

Migration To evaluate the live migration and session migration features of our BRAS, we set-up the testbed shown in figure 3. The testbed consists of two identical servers to host the BRAS, and a third server acting as traffic generator and live migration/session migration orchestrator.

We boot a BRAS on the first server, install 14K sessions, and start generating traffic at 10Gbit/s rate with maximum-sized packets. While the traffic is forwarded, we perform either live migration or session migration, moving the BRAS to the second server. During the process, we measure the downtime, that is, the time during which the traffic is not being forwarded.

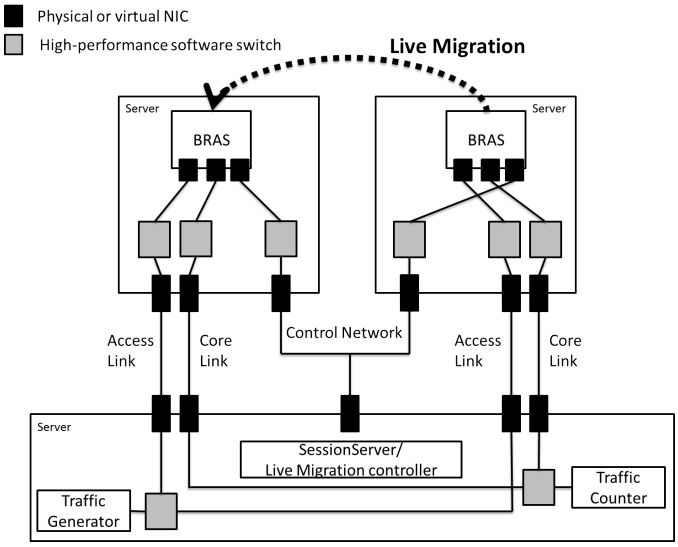

Fig. 3. Live and session migration testbed set-up. Two servers are used to host BRAS instances, while a third one is in charge of traffic generation, measurements, and of running the *SessionServer* that orchestrates session migration.

In the live migration case, the migration trigger consists of simply using Xen's `xl migrate` command to migrate the BRAS VM. The results in figure 4 (the line labeled "Live migration") show that the offered throughput has an effect on the measured downtime. This is because live migration performs an incremental copy of the VM memory. This incremental copy tries to minimize the amount of memory to transfer before shutting down the VM and moving it to the new location. Handling a high number of network packets requires the VM to use memory more frequently (e.g., to allocate new buffers), making the incremental copy process longer and less

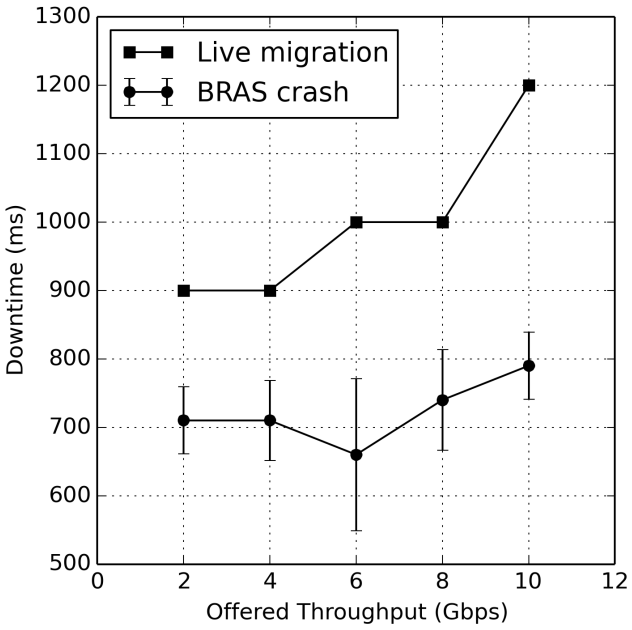

Fig. 4. Downtime observed when live migrating the BRAS and in the event of a (simulated) crash.

efficient. Still, the measured downtime is a rather reasonable 0.9-1.2 seconds depending on load. It is worth pointing out that this is the extreme case of a single BRAS VM handling 10Gb/s of traffic; in practice, it would certainly be possible to run multiple VMs and split the load between, thus reducing downtime.

In the session migration case, we can easily achieve zero-downtime, since it is possible to run two BRAS VMs in parallel. Once the second VM has been updated with the sessions' state, it is sufficient to redirect the traffic to it and turn off the first VM. Hence, for the session migration we decided to measure the downtime in the case of a (simulated) BRAS failure. We turn off the BRAS VM on the first server, and only after that do we boot a new BRAS VM on the second server and install all the sessions using the BRAS interface; this approach simulates a reactive remediation strategy to the BRAS failure.

The results (the "BRAS crash" line in figure 4) show that with session migration we can reactively recuperate from an unexpected failure in under 800 milliseconds, even though we are creating a new VM in the process.

V. Related Work

In recent years, the research community has spent considerable effort towards implementing and evaluating high performance, software-based routers. In [2], the authors present a scalable software router consisting of several commodity servers, while [3] focuses on virtualizing the forwarding function. More generally, the trend towards virtualizing network functions, as embodied by the ETSI Network Function Virtualization Industry Specification Group [5], while somewhat recent, is not new. However, to the best of our knowledge, this is the first work specifically targeting the virtualization of software BRASes and their related functions, and to include a high performance prototype and evaluation of such a system.

Finally, ClickOS, the platform on which our prototype is based, is presented in detail in [7], while an early report of the work described in this paper was presented in [1]. This work differs from that previous one by extending the design, by providing a detailed performance evaluation, and by introducing advanced features such as BRAS live and session migration.

VI. Conclusion

In this paper we presented the design, implementation and evaluation of a virtualized software BRAS. Our prototype runs on inexpensive commodity servers and is able to forward millions of packets per second using a single CPU core. Further, the BRAS supports transparent session migration (by shifting the BRAS VM) and fine-grained migration and failure recovery.

We believe that a high performance, virtualized, fast booting BRAS opens up a number of possibilities for network operators, including distribution of BRAS functions, reactively scaling out as a response to load (in millisecond time scales), and personalized BRAS instances, to name a few; exploring these and other applications is the subject of future work.

References

[1] Bifulco, R., Dietz, T., Huici, F., Ahmed, M., Martins, J., Niccolini, S., and Kolbe, H.-J. Rethinking access networks with high performance virtual software brases. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on* (Oct 2013), pp. 7–12.

[2] Dobrescu, M., Egi, N., Argyraki, K., Chun, B.-G., Fall, K., Iannaccone, G., Knies, A., Manesh, M., and Ratnasamy, S. Routebricks: exploiting parallelism to scale software routers. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 15–28.

[3] Egi, N., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., and Mathy, L. Towards high performance virtual routers on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference* (New York, NY, USA, 2008), CoNEXT '08, ACM, pp. 20:1–20:12.

[4] Fayazbakhsh, S. K., Chiang, L., Sekar, V., Yu, M., and Mogul, J. C. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr. 2014), USENIX Association, pp. 543–546.

[5] Institute, E. T. S. Industry specification group.

[6] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. The click modular router. *ACM Trans. Comput. Syst. 18*, 3 (Aug. 2000), 263–297.

[7] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., and Huici, F. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, Apr. 2014), USENIX Association, pp. 459–473.

[8] Open networking foundation. `https://www.opennetworking.org/`.

[9] AT&T, BT, CenturyLink, China Mobile, Colt, Deutusche Telekom, KDDI, NTT, Orange, Telefom Italia, Telefonica, Telstra, and Verizon. Network function virtualization - white paper.

[10] Broadband forum. Tr101v2 migration to ethernet-based dsl aggregation - issue 2, 2011.

[11] OECD. Broadband portal http://dx.doi.org/10.1787/888932398138.