Resilient Home Internet: Failure Detection in ISP Network

Technical University of Darmstadt
KOM Project, Topic 6, Winter Term 2022/2023

Régis Fayard
regis@fayard.de

Minh Thong Pham
minhthong.pham@stud.tu-darmstadt.de

***Abstract*—Virtual Broadband Network Gateways (vBNG) are a promising alternative to hardware-based BNGs for providing internet access to subscribers of Internet Service Providers (ISPs). However, vBNGs raise new challenges, especially in terms of ensuring high reliability and continuous internet connectivity in the event of a server failure. To address this, an architecture was designed where a controller migrates sessions to a redundant BNG in hot-standby. As soon as a failure is detected, the redundant BNG can take over and the subscriber connection remains uninterrupted. A proof-of-concept was implemented where sessions could be migrated successfully between PPPoE servers and server failures could be detected. The results suggest that implementing redundancy may be an important factor to increase the resilience of BNG and to assure a high availability of its services.**

I. Introduction and Motivation

The Broadband Network Gateway (BNG) which is also referred to as Broadband Remote Access Server (BRAS) is crucial to provide internet access to customer of Internet Service Providers (ISP). It connects residential gateways to the internet, see Figure 1. BNGs functions include authorization, authentication, accounting, managing subscriber sessions, aggregating traffic and enforcing policies [1], [2]. Traditionally, BNGs are usually implemented using dedicated hardware, which leads to problems like vendor lock-in, high acquisition and maintenance cost [1], [3]. Furthermore, BNG present a single point of failure, meaning that in the event of a failure, thousands of subscribers may lose their internet connection [1]. This can cause huge damage to companies as nowadays almost every transaction or communication is done via the internet.

Hence, the trend is to virtualize BNGs in software by utilizing promising concepts of software-defined networks (SDN) and network function virtualization (NFV). The advantages are that virtualized BNGs can operate with less expensive hardware, functions can be added, and they are better scalable. However, the deployment of virtual Broadband Network Gateways (vBNG) poses new challenges, especially when it comes to ensuring a high reliability and continuous internet connectivity in the event of a server failure. To improve the resilience of vBNG, a redundant system can be created by having a backup vBNG in standby, ready to take over in the event of a failure. One of the key challenges in this regard is the detection of a vBNG failure and the efficient migration of active sessions from a failed server to a backup server. However, only a few works have addressed this, especially related to BNG (related resarch mostly addressed BRAS).

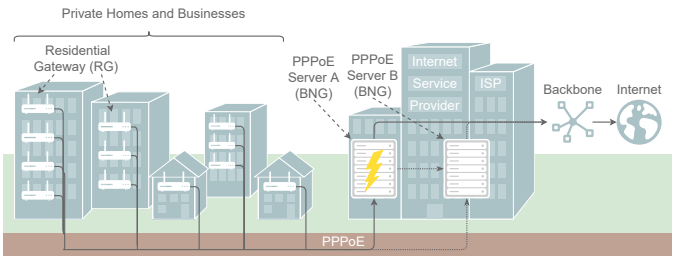

Fig. 1: Link between residential gateways and BNGs. A failure at BNG A can lead to a lot of connection losses if no resistant solution is implemented.

Therefore, the objective of this research is to investigate how the failure of a vBNG can be detected, and how active sessions can be migrated to a backup vBNG. The contributions of this work are as follows:

- Describing and comparing three ways to detect a vBNG failure.
- Presenting a design and implementation of a controller with an application programming interface (API), being able to detect a failing vBNG and migrate sessions to a backup vBNG. The controller enables the backup vBNG to be in hot-standby to ensure service continuity with little or no downtime.
- Providing a web-based Graphical User Interface (GUI) that is able to monitor sessions of vBNGs and control the automatic and manual migration of sessions between vBNGs.
- Enabling load balancing between two vBNGs triggered by the web-based GUI, by migrating sessions from one to the other.

In this paper, we first present related work and background regarding BNGs and session migration in Section II. In Section III the design of our solution is presented, followed by the implementation described in Section IV. Section V evaluates the implemented solution and finally a discussion and conclusion is given in Section VI.

II. Background and Related Work

This section provides some fundamental concepts to enhance the readability of the paper. Furthermore, an overview of the previous relevant research will be presented.

A. Developing virtual BNGs

In order to maintain consistency and avoid confusion, this paper utilizes the newer terminology of Broadband Network Gateway (BNG) rather than the traditional term Broadband Remote Access Server (BRAS). This terminology will be consistently used throughout the paper, even if referring to some "older" related work using the term BRAS.

In 2014 Intel published a white paper explaining how to build a virtual BNG and evaluated its performance [4]. The researchers concluded that the software architecture and the configuration options need to be carefully thought through, to assure a comparable high throughput as a BNG running on bare metal. In later work [5], Intel reanalysed the design of a virtual BNG and pointed out the importance of performance scalability. The virtual BNG should only use as many resources as the current number of active users requires, and should therefore be able to scale up as well as down. To use CPU resources more effectively, they use separated uplink and downlink processing.

Martins et al. developed ClickOS a high-performance, virtualized middlebox platform and implemented a BNG using it [6]. Another open source BNG prototype was built in [7] by using ACCEL-PPP [8] as PPPoE server, xDPd [9] as virtual switch and FreeRADIUS [10] for authentication

In [1], to address the limitation of a hardware-based BNG, the authors implemented and designed a virtual BNG using ClickOS (based on the work of [6]). Recognizing that transferring sessions to another BNG – such as for load balancing or reliability reasons – is complex, they implemented two ways to migrate sessions: transparent migration and fine-grained migration. Transparent migration consists of moving the entire virtual machine (VM) with its states. Because their virtual BNG runs in a minimalistic VM (MiniOS) instead of full-fledged Linux, the downtime is limited. However, sessions cannot be migrated individually, but only all at once. Another disadvantage is that in the event of a BNG failure, all sessions would be lost. The second solution is the fine-grained migration, where sessions are read from one BNG VM, then copied to another BNG VM. In their evaluation, the transparent migration had a downtime of 900 to 1200 milliseconds. The fine-grained migration was tested in two ways. First by having two BNG VMs running in parallel and then migrating sessions from one to the other with zero downtime. Secondly, by turning off one BNG VM and then starting a new BNG VM where they copied the sessions to. In this scenario, the downtime was 800 milliseconds. This paper did not discuss how sessions can be retrieved if a BNG VM fails and can no longer be read. We want to address this in our work.

B. Migration

In the field of Network Function Virtualization (NFV) the concept of migrating functions while they are still active is known as live migration and has been extensively researched [11], [12]. Live migration is often archived by deploying a new virtual machine (VM) and transferring the memory disk state of the running VM to it. Inevitably, live migration causes some downtime and high traffic overhead during the migration [11], [12].

In the context of 5G radio access network (RAN), to cope with the already named limitations of live migration, Alba et al. introduced the so-called replication-based approach [12]. In this replication-based approach, the state of the old set of function is transferred to the new one. Compared to migrating a whole instance, it is faster and results in less traffic overhead [12]. Another approach was proposed in [13], where Linux Containers (LXC) were used to perform lightweight live migrations in 5G networks.

C. Point-to-Point Protocol over Ethernet (PPPoE)

Point-to-Point Protocol over Ethernet (PPPoE) is a network protocol that encapsulates PPP packets over Ethernet [14], [15]. Internet service providers (ISPs) use PPPoE to provide customers or subscribers access to the internet [16]. In this use case, the residential gateway of the subscriber is a PPPoE client and the BNG the PPPoE server.

PPPoE has two stages, the discovery stage also called PPPoE handshake (see Figure 2) and the session stage [15]. The discovery stage is necessary so that the PPPoE client discovers the MAC address of all accessible PPPoE servers and then selects one to use [15].

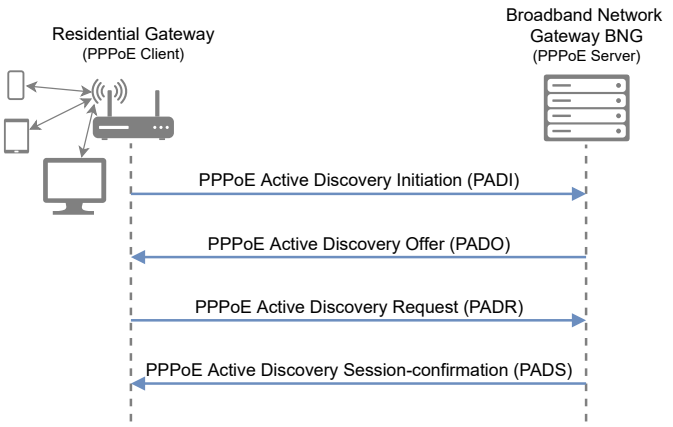

Fig. 2: PPPoE Discovery handshake between a residential gateway and a BNG.

The discovery stage consists of four steps, where the following packets are exchanged: [15]

1) First, the PPPoE clients starts by broadcasting a PPPoE Active Discovery Initiation (PADI) packet, in order to find a PPPoE server, see Figure 2.
2) All the PPPoE servers that received the PADI reply by sending a PPPoE Active Discovery Offer (PADO) packet

via unicast. The PADO contains the server's name and some additional service tags.

3) If the PPPoE client received multiple PADO, it chooses one based on the name or services offerd. Then it sends a PPPoE Active Discovery Request (PADR) packet to the chosen one.
4) When the PPPoE server receives a PADR, it generates a unique session ID. It then confirms the connection, by sending a PPPoE Active Discovery Session-confirmation (PADS) packet, which contains the session ID.

After a successful discovery stage, the session stage begins. From now on, all messages send will include the session ID in the PPPoE header. The Ethernet frame used for PPPoE has a maximum payload of 1500 bytes. Of them, 6 bytes are used for the PPPoE header and 2 bytes for the PPP Protocol ID. The Maximum-Receive-Unit (MRU) of a PPPoE server is 1492 bytes. [15]

To terminate the established session, the PPPoE client or the PPPoE server can send at anytime a PPPoE Active Discovery Terminate (PADT) packet. When a PADT is received, no more PPP traffic is permitted to be sent using that session [15].

III. Design

The aim of the following concept design, is to improve the reliability of a virtual BNG (PPPoE server). PPPoE clients such as residential gateways connect to the PPPoE server in order to access the internet. Each successful connection of a PPPoE client is saved by the server as a session. The core of the concept is the controller. It has the task of detecting whether the server is failing. To be prepared for a possible server failure, the controller proactively migrates the active sessions of the primary PPPoE server to the backup PPPoE server. As soon as a failure is detected by the controller, it signals the switch that the incoming packets of the clients should no longer be forwarded to the primary PPPoE server but rather to the backup PPPoE server, see Figure 3. The advantage is that the clients do not notice the failure of the PPPoE server and do not need to reconnect to a PPPoE server. This is possible because the clients' active sessions are continued on the backup PPPoE server.

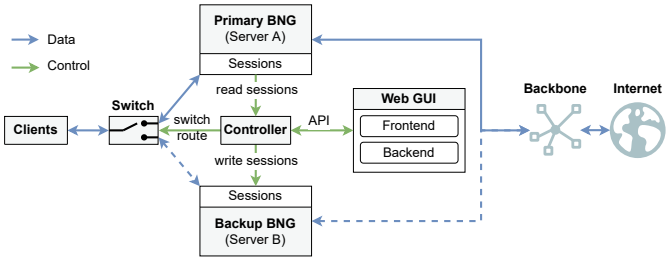

Fig. 3: Controller with web GUI to migrate sessions from Server A to Server B in the event of a failure of Server A.

One factor to consider when adopting this concept is where to place the virtual BNGs. The BNGs and the controller could be run in the same server unit. This would have a positive effect on performance because of the low latency. However, in the event of a hardware failure of the server, for example, both the primary and the backup BNG would be affected simultaneously, and the system would have missed its purpose. The other extreme would be to run the primary BNG, the backup BNG and the controller in three separate data centres. This way, it would be very unlikely that all three would fail at the same time. However, the distance between the three components would have a negative impact on latency and performance. A strategic balance between performance and resilience would have to be determined.

The concept presented can also be extended to more than just two vBNGs. In this way, load balancing can be performed, where the controller distributes the clients between several BNGs depending on their workload.

With an application programming interface (API) provided by the controller, the servers and controller can be monitored and controlled. A web-based Graphical User Interface (GUI) can be used by the people responsible for the vBNG as a user-friendly interface to interact with the controller and its API. The GUI displays the current state and the sessions of each PPPoE server, whereas sessions can be migrated to another server, e.g., for load balancing. Moreover, automatic session migration in event of failure can be enabled or disabled. As the controller and the GUI have been deliberately separated from each other with an API, the controller continues to run if the GUI crashes.

IV. Implementation

This section explains the implementation of the concept and the software utilized for the PPPoE server, PPPoE client, controller, and Web GUI.

A. PPPoE Server

As a virtual BNG (PPPoE server), ACCEL-PPP is used. It is a high performance PPTP/L2TP/SSTP/PPPoE/IPoE server that runs on Linux [8]. For this implementation, the server's PPPoE-function is particularly relevant. With the official release of ACCEL-PPP, sessions can only be read from the server. For migrating sessions, it is necessary to be able to write them on the server. Therefore, a version of ACCEL-PPP modified by Fridolin Siegmund[1] was used, which can also write sessions. In this project, we run two instances of the modified ACCEL-PPP on an Ubuntu 22.04 LTS[2] machine. Each instance runs within a separate namespace, permitting isolation of the two instances and traffic routing to specific instances, see Figure 4. The first instance is the primary PPPoE server, namely Server A, whereas the second instance is the backup PPPoE server, namely Server B.

To read and write sessions on ACCEL-PPP the provided command line tool accel-cmd is used. When accel-cmd is executed through the command line, it establishes a connection over TCP on port 2000 with the ACCEL-PPP instances to fulfil the request [17]. The accel-cmd command to read sessions,

[1]Multimedia Communications Lab, Technische Universität Darmstadt
[2]www.ubuntu.com

returns all the sessions of the ACCEL-PPP instance as JSON. Writing sessions to ACCEL-PPP comes with some limitations, as ACCEL-PPP must be initialized with at least one session in order for new sessions to be successfully written to. To overcome this issue, we first check if there are already sessions before migrating or writing new sessions to ACCEL-PPP. If no sessions are present, a temporary PPPoE connection is established before writing a session on the server.

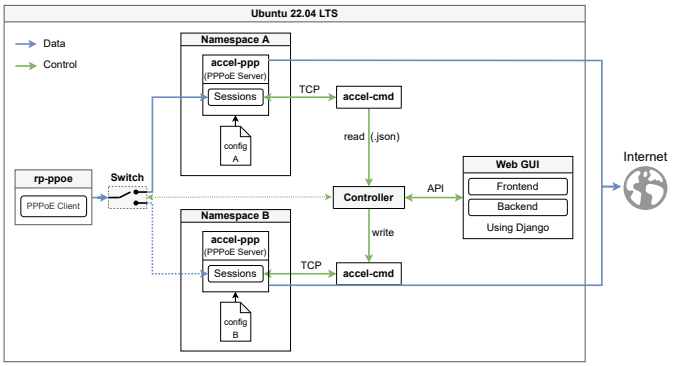

Fig. 4: Components used in the implementation and how they interact with one another.

B. PPPoE Client

RP-PPPoE is used [18] as a PPPoE client that connects to the PPPoE server. RP-PPPoE can establish a connection with the PPPoE server (PPPoE discovery) and authenticate itself after the discovery phase. This is necessary for successfully storing its session on the server. Due to the simplicity of the set-up, RP-PPPoE was chosen instead of other PPPoE clients, as for example pfSense[3] or OPNsense[4]. However, an implementation with these PPPoE clients would also be possible.

C. Controller

The controller which detects a PPPoE server fault and migrates sessions to a backup server is implemented in python. The controller runs with two parallel processes. The first process runs the control loop, which is an infinite loop that periodically checks the availability of the two PPPoE servers. This process is also responsible for the automatic migration of sessions from Server A to Server B. The second process handles the API server that will start a new thread for each API connection, which will handle the requests and respond.

Separating the control loop and API in two processes, instead of one process running two threads, has the benefit of not having race conditions and enforcing more isolation between the API and the control loop. See Figure 5 representing the two processes as flow chart. Thereby, it reduces the chance that a fault in the API, affects the process with the control loop and strengthens the resilience. Each process has its own local variable and memory space, the only shared information

[3]www.pfsense.org
[4]www.opnsense.org

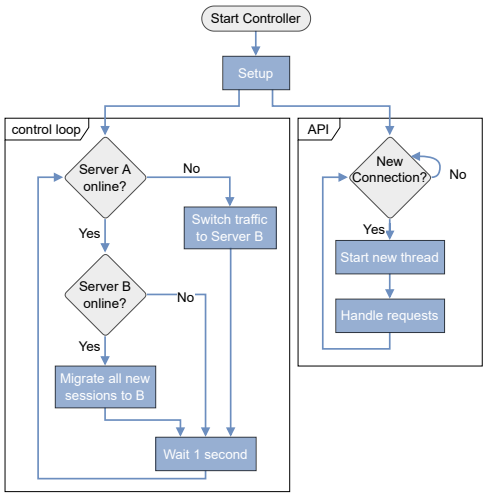

Fig. 5: Simplified representation of the two processes run by the Controller

after the initial setup, is a synchronized shared boolean, so that the control loop can be instructed via the API whether sessions should be migrated automatically or not.

Three approaches were analysed to implement the failure detection of PPPoE servers before choosing and implementing the best approach. These three approaches are given and discussed below:

1) Ping the server running ACCEL-PPP and wait for a response. If the server is running and the connection to it is working, it will send a response to the ping request.
2) ACCEL-PPP periodically sends a packet to inform the controller that it is still running.
3) The controller periodically requests to read the sessions inside ACCEL-PPP. It will only get a response if ACCEL-PPP is running properly and the connection to it is working, see Figure 6.

One limitation of the first approach is that a ping response only indicates the reachability of the server and does not provide information on the proper functioning of ACCEL-PPP. Hence, it was discarded. The second and the third approach do not have this limitation. For the third (and first) approach, packets are sent two-ways. Compared to the second approach where packets are only sent one-way, the third approach leads to higher latency. However, to implement the second approach, ACCEL-PPP source code would have to be modified. As this is complex to implement, it is a drawback for the second approach.

The advantages and disadvantages of the three approaches are summarized in Table I. The third approach was chosen as it has the most arguments in its favour.

Furthermore, for the third approach, the sessions are read from the server. Reading the sessions can be used for detecting a server failure as well as for migrating sessions to the backup server, which is convenient. More details on this are given later.

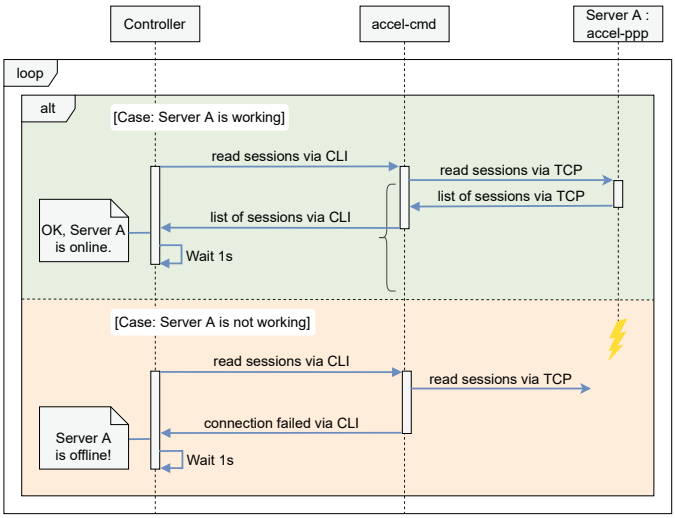

Fig. 6: Detecting, if ACCEL-PPP and the connection are working properly by periodically requesting to reading its session.

Approach	1	2	3
Connection	Two-Way	One-Way	Two-Way
ACCEL-PPP modification needed	no	yes	no
Detects connection failure	yes	yes	yes
Detects ACCEL-PPP failure	no	yes	yes

TABLE I: Comparison of the three approaches to detect a PPPoE server failure.

To minimize the interruption time in the event of a failure, it was opted to have the backup server in hot-standby. This means that the backup server is always in sync with the primary server and is ready to take over in the event of a failure. In cold-standby, the server must first be started and then loaded with the sessions, which costs valuable time. To keep the backup server in sync for the hot-standby, the sessions are periodically (every second) read from Server A and migrate to Server B. The sessions that were already read when checking if the server is available are now used to be migrated to the backup server.

By implementing an API for the controller, the controller and PPPoE servers are monitored remotely. For this, the python library RPyC (Remote Python Call) was utilized [19]. With RPyC, the function of the controller can be called remotely via TCP (Transmission Control Protocol). The implementation of an API results in the following features:

- Start and stop the primary and backup PPPoE servers (Server A and B).
- Check the availability of Server A and B.
- Enable and disable the automatic migration of sessions from Server A to B (needed for the hot-standby configuration of Server B).
- Read and write sessions on Server A and B.
- Migrate all sessions from one server to another server.
- Write dummy sessions to Server A and Server B for testing and debugging purposes.

D. Web GUI

The web-based GUI's goal is to make it user-friendly to interact with the controller and its API. For the backend of the GUI the framework Django version 4.1 is utilized [20], whilst for the frontend HTML and Bootstrap 5.3 is used [21].

The GUI consists of two parts, in the upper part buttons can be used to control the ACCEL-PPP servers and the controller, see Figure 7. For each API features described before, the GUI provides buttons. In the lower part, all sessions of each server including details are listed. Sessions can be migrated individually or all at once to another server. Badges indicate if the server or controller are reachable (online). The GUI uses server-side rendering, so needs to be refreshed to display the latest data.

V. Evaluation

A series of tests are performed to evaluate the proof-of-concept described in the previous section. In particular, the behaviour of the controller and the GUI are tested and the duration of several operations is measured. The tests are performed on a virtual machine running Ubuntu 22.04 LTS with 6 GB of memory and 1 CPU core.

The controller, API and GUI are tested by having a PPPoE client connect to the PPPoE Server A. According to the tests, the session of the client can be successfully read by the controller and displayed in the GUI. Furthermore, the different ways to migrate sessions are tested. The three ways, migrating a single session, migrating all sessions of a server, and the automatic migration of new sessions to the backup server, worked well.

According to the tests, the controls to start the ACCEL-PPP servers via the GUI (or the API) do not always function reliably. For one test setup (VM) it works flawlessly, for another test setup it doesn't work. Consequently, there seems to be a programming bug. Starting the ACCEL-PPP server without the GUI does not cause any problems.

In another test, it is examined how many sessions can be written and read on an ACCEL-PPP instance. Four dummy sessions can be written and then read without any problems. But as soon as a fifth one is written, the sessions can not be read correctly. The JSON returned by accel-cmd which should contain all sessions, is clipped and incomplete. The interface of ACCEL-PPP for reading sessions is probably responsible for this problem.

To assess the advantages of having a backup server in hot-standby versus restarting a failed server, the reboot time of ACCEL-PPP is first measured. The measurement is conducted ten times, resulting in an average of 36.6 milliseconds (SD = 12.29 ms). After ACCEL-PPP is restarted, clients would have to reconnect to the PPPoE server. Therefore, the duration for a PPPoE discovery handshake between a client and ACCEL-PPP is measured. This is accomplished by analysing the duration between the timestamps of a PADI and a PADS from a TCP dump. The duration is measured six times, resulting in an average of 3.16 milliseconds (SD = 3.81 ms). If the authentication process (using CHAP) and IP address

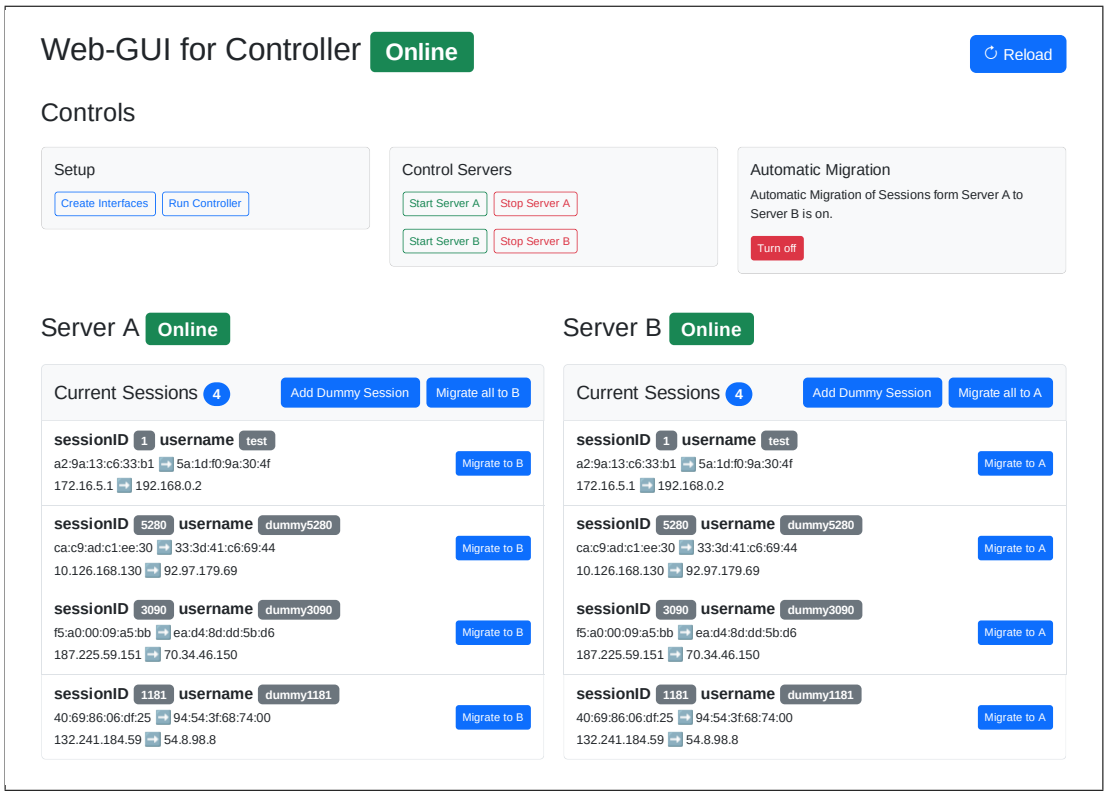

Fig. 7: Web GUI to interact with the controller.

configuration (using IPCP) are considered together with the PPPoE discovery handshake, the average duration increases significantly to 5400 milliseconds (SD = 5528 ms). Moreover, it is important to note that in this test case the client and server are running on the same virtual machine, which does not reflect a realistic use case where the client and server are located at different locations, separated by multiple kilometres. This can result in much higher Round Trip Times (RTT) in real-world scenarios.

The time the controller takes to migrate a session from one server to the other is measured by the controller. Migrating four sessions in five tests gives an average of 176.3 milliseconds (SD = 2.8 ms) per migrated session. Due to time constraints, the virtual switch (see Figure 3), responsible for re-routing traffic to the backup server after the failure of the primary server, is not implemented. Therefore, it is not possible to fully evaluate the migration from one server to another.

All measurements of the evaluation are summarized in Table II.

	Average	SD
Reboot ACCEL-PPP (PPPoE server)	36.60 ms	12.29 ms
PPoE discovery handshake	3.16 ms	3.81 ms
PPPoE discovery handshake + authentication + IP address configuration	5400 ms	5528 ms
Migrate a session from one ACCEL-PPP server to the other	176.3 ms	2.8 ms

TABLE II: Measured durations to evaluate the proof-of-concept.

VI. Discussion, Conclusion, and Future Work

In this section, the concept and the results are discussed, conclusions are drawn and potential future research is outlined.

A. Discussion

This paper presents an architecture for enhancing the resilience of BNGs, which has the potential to significantly improve network availability for subscribers in the event of a BNG failure. The implementation of a proof-of-concept demonstrated the feasibility to automatically migrate sessions to a backup PPPoE server, as soon as they are established on the primary PPPoE server, and thus making it possible to keep a backup PPPoE server in hot-standby.

According to the results of the evaluation of the proof-of-concept, a second PPPoE in hot standby is beneficial. In the scenario where a PPPoE server has a failure and restarting it would be sufficient to get it working again, all active subscribers would have lost their session and would be required to re-establish a session with the PPPoE server. Our experiment revealed that the duration required for the PPPoE discovery handshake, authentication, and IP address configuration that is necessary for subscriber reconnection is significant, at an average of 5400 milliseconds. Having a backup server that already contains the sessions prevents subscribers from having to reconnect and thus improve the

quality of the service.

In the following, the necessity of having the backup server in a hot-standby versus a cold-standby configuration is discussed. With cold-standby, once the primary server has a failure, the backup server is started and all sessions are migrated from a cache to it. The duration measured showed that in average the PPPoE server needs 36.6 milliseconds to start and the migration of one session took 176.3 milliseconds. Having potentially thousands of sessions to migrate, would lead to considerably high interruption times for subscribers. This highlights the advantage of having the backup server in hot-standby and constantly migrating newly established sessions to it. This ensures that the backup server is immediately able to operate in the event of a failure.

A limitation of our concept is that during the interval between the failure of the primary server and the detection of the failure by the controller, followed by the re-routing of traffic to the backup server, packets may be dropped or lost. The rate of the controller checking for a server failure was set to one second for the prototype, this could be further optimized by increasing the rate, and hence reducing the number of packets lost. However, this comes with a trade-off, as a higher rate could negatively impact the performance of both the controller and the PPPoE server. Further analysis is necessary to figure out the appropriate balance. The extent of lost packets could not be further analyzed, as the switch required for this was not implemented due to time constraints.

A web-based graphical user interface (GUI) was implemented to facilitate the control of PPPoE servers and monitoring of sessions. This GUI interacts with the controller via an API. However, the GUI must be manually refreshed to display the most recent information.

While the failure detection and the migration of a few sessions worked well in our proof-of-concept, further research is still necessary to study its scalability and extend the scope from a PPPoE server to a fully-fledged BNG. Nevertheless, our findings provide new insights on how to improve the reliability of BNGs and suggest that implementing redundancy may be an important factor for making BNG more resilient and to assure a high availability of its services.

B. Conclusion

In this paper, we presented a concept of a redundant virtual BNG system, where in the event of a failure of a primary BNG, a backup BNG in hot-standby assures the service continuity of all the active subscriber sessions. A proof-of-concept was accomplished by implementing a controller which continuously reads the sessions from the primary PPPoE server (BNG) and migrates them to the backup PPPoE server (BNG). By utilizing this approach, the traffic can be quickly re-routed as soon as the controller detects a server failure, thereby minimizing the potential of subscriber downtime. A web-based graphical user interface (GUI) was developed to facilitate the handling of the controller and simplify the monitoring of the PPPoE servers. The GUI communicates with the controller through an API.

Our proposed solution holds promise as an approach to improve the resilience of virtual BNGs and thereby provides a high level of service availability for customers.

C. Future Work

While our study has advanced our understanding on how to improve the resilience of virtual BNGs, there is still a need for further research to elaborate it and to address the limitations.

The proof-of-concept could be further developed, for example, by implementing the missing virtual switch functioning as a switchable Layer 2 access network. This would allow more extensive evaluations to be conducted.

For the controller, the following approach is worth comparing with the implemented one. Instead of the controller running in two separate processes (one for the API and one for failure detection and migration), they could also be run in two separate threads. This would allow the sessions retrieved during failure detection to be cached, so that the API could output them in response to requests. This way, the API would not have to read sessions from the server every time it receives a request. The queries to the PPPoE server could thus be reduced significantly and therefore increase its performance.

Furthermore, the Web GUI could be improved with client-side rendering to update automatically if new data is available or the API could be used for other research, e.g. integrating it in other systems.

As an alternative to migrating sessions from one PPPoE server to the other to archive hot-standby, the sessions could instead be saved externally. It would be accessible from both the primary and backup PPPoE server. Hence, no sessions migration from one server to the other would be necessary. In future research, it could be beneficial to conduct a comparative analysis of this approach with the approach presented in this paper.

REFERENCES

[1] T. Dietz, R. Bifulco, F. Manco, J. Martins, H.-J. Kolbe, and F. Huici, "Enhancing the BRAS through virtualization," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. London, United Kingdom: IEEE, Apr. 2015, pp. 1–5.

[2] "Broadband Network Gateway: A new access portal for the web," Jun. 2013. [Online]. Available: https://www.capacitymedia.com/article/29ot5qo2lye3ttjhncxkw/news/broadband-network-gateway-a-new-access-portal-for-the-web

[3] Y. Zhou, W. Li, J. Bi, Y. Ma, C. Sun, and P. Chen, "hBRAS: Towards high-performance broadband remote access services with programmable switches," 2018.

[4] Intel Corporation, "Network Function Virtualization: Virtualized BRAS with Linux* and Intel® Architecture," 2014.

[5] P. Connolly, A. Duignan, and S. Hiremath, "Re-Architecting the Broadband Network Gateway (BNG) in a Network Functions Virtualization (NFV) and Cloud Native World," Oct. 2021. [Online]. Available: https://networkbuilders.intel.com/solutionslibrary/re-architecting-the-broadband-network-gateway-bng-in-a-network-functions-virtualization-nfv-and-cloud-native-world

[6] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "{ClickOS} and the Art of Network Function Virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 459–473.

[7] BISDN GmbH, "Carrier solution: BNG Gateway."

[8] "Accel PPP," Accel PPP Inc, 2022. [Online]. Available: https://github.com/accel-ppp

[9] "xDPd," BISDN GmbH, Dec. 2022. [Online]. Available: https://github.com/bisdn/xdpd
[10] "FreeRADIUS," FreeRADIUS project, Jan. 2023. [Online]. Available: https://github.com/FreeRADIUS/freeradius-server
[11] W. Cerroni and F. Callegati, "Live migration of virtual network functions in cloud-based edge networks," in *2014 IEEE International Conference on Communications (ICC)*, Jun. 2014, pp. 2963–2968.
[12] A. M. Alba, J. H. G. Velasquez, and W. Kellerer, "An adaptive functional split in 5G networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Paris, France: IEEE, Apr. 2019, pp. 410–416.
[13] R. A. Addad, D. L. Cadette Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Towards a Fast Service Migration in 5G," in *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct. 2018, pp. 1–6.
[14] W. A. Simpson, "The Point-to-Point Protocol (PPP)," Internet Engineering Task Force, Request for Comments RFC 1661, Jul. 1994.
[15] D. Carrel, J. Evarts, K. Lidl, L. A. Mamakos, D. Simone, and R. Wheeler, "A method for transmitting PPP over ethernet (PPPoE)," RFC 2516, Feb. 1999.
[16] F. Liu, T. Xie, Y. Feng, and D. Feng, "On the security of PPPoE network," *Security and Communication Networks*, vol. 5, no. 10, pp. 1159–1168, 2012.
[17] Accel PPP Inc, "Control features — accel-ppp latest documentation," 2023. [Online]. Available: https://docs.accel-ppp.org/en/latest/guides/control_features.html#accel-cmd
[18] D. Skoll, "Rp-pppoe," 2022. [Online]. Available: https://salsa.debian.org/dskoll/rp-pppoe
[19] "RPyC," tomerfiliba-org, Jan. 2023. [Online]. Available: https://github.com/tomerfiliba-org/rpyc
[20] "Django," Django Software Foundation, 2023. [Online]. Available: https://www.djangoproject.com/
[21] "Bootstrap," Bootstrap Team, 2023. [Online]. Available: https://github.com/twbs/bootstrap