Overlay Network Based Optimization of Data Flows in Large Scale Client-Server-based Game Architectures for Deployment on Cloud Platforms

Peter Quax, Wouter Vanmontfort, Robin Marx, Maarten Wijnants, Wim Lamotte
Expertise Centre for Digital Media – tUL – IBBT
Wetenschapspark 2
3590 Diepenbeek
Belgium
e-mail: {peter.quax, wouter.vanmontfort, maarten.wijnants, wim.lamotte}@uhasselt.be

Paper type

Full paper

Abstract

In this overview paper, we present results that were obtained by combining a specific architectural design for Networked Virtual Environments (NVEs) with a generic optimization infrastructure for content streaming. Although the two are –at first sight – targeting unrelated application areas, it turns out that the application knowledge available in the NVE architecture can effectively be used to instruct the adaptation infrastructure to change the way in which data is delivered to the end-user. The end-effect of these optimizations is a fine-grained ability for the combined architecture to deliver the required NVE content to the end-user while optimizing bandwidth usage. Such optimizations are very useful in mobile setups, as the available bandwidth fluctuates significantly over time. The resulting architecture can be deployed in a very dynamic way, including on cloud platforms. An experiment is presented that shows the viability of the ideas discussed in the paper.

Keywords

Networked Virtual Environments, Scalability, Cloud Architectures, Proxy architectures

1. Introduction

Networked Virtual Environments are the core technology behind popular Massive Multiplayer On-line Games such as World of Warcraft, EVE Online and Guild Wars 2. Although a number of commercially successful examples currently exist, there is a lot of room for improvement and need for research into the uptake of this technology by smaller developers and publishers and deployment on mobile networks. A number of issues can hinder the practical deployment for these target groups. First of all, the investment needed to get a basic infrastructure up and running is prohibitive, unless users are charged for access. Secondly, the game developer or publisher needs to pre-determine the likely degree of popularity for the application. If this is not done very carefully, it is likely that the server capacity will be either overdimensioned (which wastes resources and money) or underdimensioned (leading to interaction latency and users losing interest in the application). On an architectural level, the system needs to be developed in such a way that a number of possible usage scenarios can be supported by the same system, facilitating re-use of resources and code (software).
A few years ago, researchers at the Expertise Centre for Digital Media developed ALVIC-NG, a generic architecture that is flexible in its support for applications, scales well with a growing number of users and which is unique in the way it channels the message flow. Where classic examples have a two-tier approach (clients connecting directly to virtual world servers), ALVIC-NG proposes a three-

tier architecture, introducing a layer of proxy servers in-between servers and users. By inspecting the traffic flow, these proxy servers can streamline the data flows, perform caching where possible and reduce the latency experienced by end-users (which is critical to their interest in the application).
The role of the proxies in ALVIC-NG is however still quite traditional. Decisions can be made on which packets to pass or drop based on several parameters for each individual client. However, the proxies in the original proposed architecture could not take into account all semantics of the virtual world, as they could not handle the data processing associated with this work. By optimizing the processing and by extending the functionality of the original ALVIC-NG proxies with network and application intelligence (insights available through other related work at the EDM), the proxies can now adjust their workings based on the state of the virtual world, the intentions of the users and the available bandwidth capacity. In this paper, it will be shown that these optimizations and extensions lead to a more flexible system that is able to adjust itself to various contexts of use. In practice, this is demonstrated by studying the behavior of the architecture when applied to different game or application genres. In this paper, focus will be on a single scenario.
The end-effect is a more fine-grained control over bandwidth usage, which will facilitate deployment of the same application or game onto a heterogeneous set of devices and networks.

2. Related Work

For related work on the architectural design of ALVIC-NG, the interested reader is referred to our earlier work[1]. In this section, we will focus on the similarities between the extended architecture and existing examples in real-world deployments. The proxy servers in ALVIC-NG are intended to function like the Edge servers which are in use in a Content Distribution Network (such as Akamai[2] or EdgeCast). While the current 'state' of the world, needed by the end-user is always available in the back-end (on virtual world servers that are in this work referred to as Logic Servers), the proxies compose a layer in-between these parties. This way, an overlay network is formed that is (at least in theory) better able to deal with bandwidth estimation, topology changes and scalability requirements than a system based on network layer 3 information and routes.
The use of such an advanced scheme is however still not an attractive proposition for small developers and publishers in the games market, as it does not, by itself, solve the issue of investment in dedicated hardware. By introducing the third tier in ALVIC-NG however, responsibilities are divided between server entities, and some of these can even be run on popular cloud platforms. Due to the nature of the applications, only the Infrastructure as a Service (IaaS) platforms are under consideration. Of course, the likeliness that a server can be deployed on the cloud is dependent on the nature of demands posed on the server instances. Proxy servers are mainly tasked with channeling the data flows between end-users and logic servers; their main bottleneck therefore is network processing speed. Logic servers on the other hand are typically more general processing-hungry and will require a different (likely virtual in case of cloud platforms) hardware composition. The flexibility of modern cloud platforms however allow customization of virtual machines for specific purposes and are therefore ideally suited to support both proxy and logic servers in the ALVIC-NG proposed scheme. The fact that only a minimal set of servers is required by ALVIC, the fact that additional resources are easily provisioned on-demand and the fact that a customer only pays for resources consumed makes this an ideal proposition for the intended target group.

3. Architecture description

3.1 The ALVIC-NG architecture

The main entities of the architecture are shown in figure 1, represented in a set of concentric circles. At the outer perimeter the clients are shown that want to connect to the virtual world. Instead of connecting to a variety of supporting servers as is often the case in current-generation examples such as Second Life, nearly all traffic is tunneled over the client-proxy link. The proxies are responsible for handling a number of clients at the same time, and are assigned based on several properties. These may include, for example, their processing load and/or the network properties of the link between the client and the proxy (e.g. typical RTT values and/or packet loss). Proxies are assigned from a pool of available servers, managed by a centralized entity, which is also responsible for other authentication and accounting tasks.

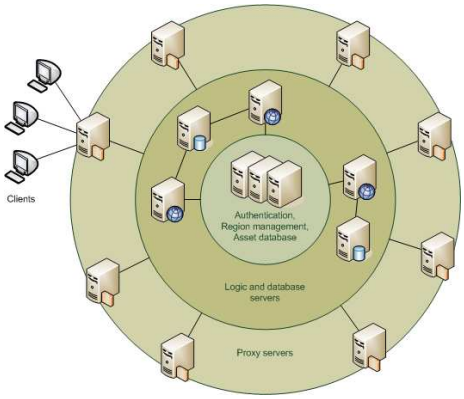

Figure : ALVIC-NG architecture overview

The entities responsible for managing parts of the world are referred to as Logic servers. They are notably different from, for example, the simulators in the Second Life[3] architecture, in the way they are assigned to geographical regions in the virtual world. Instead of using a fixed allocation scheme, as is traditionally used, a new entity is created, responsible for managing the relationship between virtual locations and Logic servers, called the Region Management (RM) system. Analogies can easily be drawn between these RM servers and the DNS system that is currently in use on the Internet. The RM system can be queried by the proxy servers to find out which region(s) is/are managed by a specific server. At the same time, the RM system is responsible for keeping track of the load on the various Logic servers. A control link is therefore established between the RM system and the individual Logic servers, over which several parameters are sent, comparable to the SNMP querying system. In case the RM system detects either a Logic server failure or an impending overload of a specific server, the Logic servers are re-assigned to remediate the problems. Possible solutions include splitting the management of a (previously) single part of the virtual world over a number of servers or transferring the complete responsibility to a new instance, e.g. in case of complete Logic server failure.
Logic servers can also be used as entities that control the behaviour of objects, such as non-playable characters (NPCs) or autonomous interactive objects such as virtual video walls. Behaviors are triggered by scripts that are assigned to specific objects. As the Logic servers are responsible for handling all objects present in a specific part of the virtual world, which will traverse the virtual word, the scripts need to be shared between all Logic servers. These scripts, together with the information regarding the visual representation of objects, are stored in asset databases.
The reason behind the introduction of the intermediary layer of proxy servers in the architecture is fourfold. First of all, it reduces the number of connections each client needs to initiate and maintain with other servers (which may lead to connection issues due to the abundant presence of firewalls and NAT gateways). Secondly, the proxies reduce the number of connections for the Logic servers, which is important if a high number of clients is to be supported on a single machine due to the overhead associated with connection tracking. Thirdly, the proxies can ’cache’ some of the data that they see flowing past, possibly reducing the response time (and load) on the Logic servers, as these servers can be assigned in such a way that they provide a better response time than the entire path between the client and the Logic server(s).

However, the latter feature is mainly applicable for non-state information such as mesh or inventory information, as the state changes are –in general– too frequent for a caching system to be useful. Finally, the proxy servers form a facilitating entity, making it possible to enhance the scalability of the entire architecture by hiding the topology changes from the end-users – this is described in detail throughout this paper.
As with any virtual environment system, persistent storage is a requirement to keep the world up and running over long periods of time. It also offers enhanced features such as roll-back capabilities in case of system failure and/or, more applicable to the virtual world scenario, in case of malevolent users that have exploited the system. Instead of using a single, high capacity database, as is typical in existing applications (e.g. EVE Online), the ALVIC-NG architecture provides a fine-grained mechanism for determining the degree of persistency that is required. In case transactions are handled that have financial repercussions (e.g. the exchange of virtual currency between users), it is likely that these transactions need to be logged and written to disk immediately, as an in-between state, where currency is ‘floating’ between users would clearly not be desirable. However, it should be clear that not all

objects and actions require an immediate storage of state to disk. This enables the ALVIC-NG architecture to retain as much state as possible in the main memory of the Logic servers, which improves both response time and the load on the database servers.

3.2 The NIProxy architecture

At any given time, there are many different types of network streams active in a typical NVE. Therefore, a robust system for managing these streams and, in this case, their bandwidth usage in particular is required. One existing framework for doing this is the Network Intelligence Proxy (NIProxy) [4], an academic research project of the Expertise Centre for Digital Media. The main goal of the NIProxy is to provide network traffic shaping abilities. It offers a means to do this by using multimedia services like on-the-fly video transcoding. The NIProxy takes the form of a non-transparent server in the network and provides an API through which applications can indicate their specific needs. By doing so, the proxy can adapt the network streams on a per-user/per-application basis, make the network more intelligent and regulate bandwidth usage so networks do not become overloaded.

An example to explain the basic idea behind the NIProxy is to think of a central live video streaming server that is streaming high-quality video to its subscribers. When a user with a mobile phone wants to watch this video stream on a mobile network, it is very unlikely that this will be possible if he subscribes directly to the high-quality video of the server. This would mean that the server would have to send multiple streams, each with a different quality and resolution, to enable mobile users to watch the video stream. However, this requires additional processing and bandwidth overhead for the central server, which will probably already have a considerable workload if it is serving popular content. The NiProxy is placed in an in-between layer and is able to perform content adaptation based on the exact requirements of the users. That way, the source only needs to output a video stream in one format, which will subsequently be transformed and/or transcoded as required. The same principle applies to all types of content and can, as will be shown in this paper, also be applied to traffic typically present in networked virtual environments.
Internally, the NIProxy uses with a tree-based architecture to structure bandwidth distribution techniques in a so-called stream hierarchy. This tree can be user-specific and can be composed of various node types (mainly exclusive nodes and weighted nodes). By evaluating the nodes in the tree, the network flows are adapted or channeled for each specific user. A generic example is shown in figure 2.

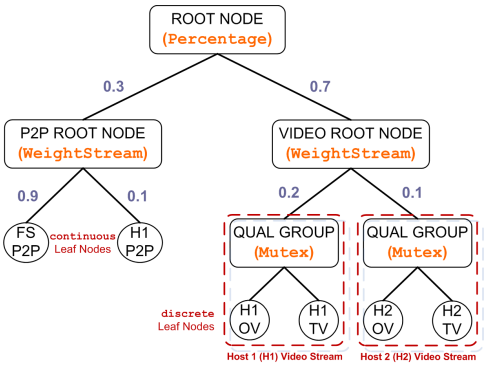

Figure 2 : Example NIProxy hierarchy

4. Combining ALVIC-NG and the NIProxy

It should be clear from the descriptions in the previous sections that a combination of ALVIC-NG and the technology in the NIProxy architectural design would benefit Networked Virtual Environments in general. By integrating the functionality of the NIProxy into the ALVIC-NG proxies, the introduction of an additional layer of indirection is avoided and the processing capabilities of the hardware can be used in an optimal way. What is needed is a way to transform the information obtained from the virtual world into instructions that are suitable for the creation of a bandwidth hierarchy tree. In this section, a representative example will be provided that shows that this is indeed possible and can lead to a reduction in bandwidth for the connected client.

In the discussion in this section, we will assume that the reader is familiar with concepts from Networked Virtual Environments and game technology such as Area of Interest determination, Level of Detail techniques and game classification. A complete overview of these techniques (and other related technologies) is provided in [5].

4.1. First Person Shooter Sniper Scenario - Overview

For reasons of clarity, this scenario is limited to three players in the world. Although one can scale the system to thousands of simultaneous users, the overview is quickly lost under such circumstances.

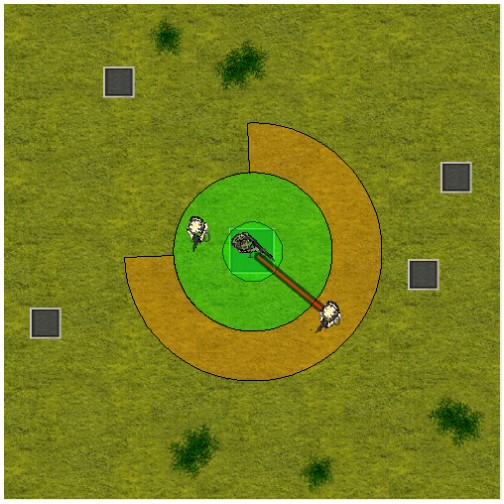

Figure 3 : Overview of Area of Interest definition in the FPS sniper scenario

Player 1 assumes the role of a sniper, sitting on top of a building in the center of the map. The 2 other players walk around the building in circles, each at a different radius from the sniper. The sniper tracks the outer player, which causes constant updates of the rotation to face the direction of the outer player. This is represented in figure 3. As the wedge shape is tied to the sniper's rotation, this will also move along. It should be noted that the above description is not a truly realistic situation, but it is kept simple to demonstrate the workings of the bandwidth shaping. At given times, the available bandwidth for player 1 will change. This simulates, for example, a handover between a 2G, a 3G and a WiFi connection for mobile stations. It will prompt the NIProxy to take over and select a new Area of Interst (AOI) definition (which it assumes will not cause the bandwidth requirements to be exceeded). The NIProxy bandwidth shaping tree is shown on the left hand side in figure 4. As can be observed, this tree is exceedingly simple. The root node is a priority node (although this does not really matter for this scenario) and it only has a single child leaf node, in charge of manipulating a single InterestDefinition.

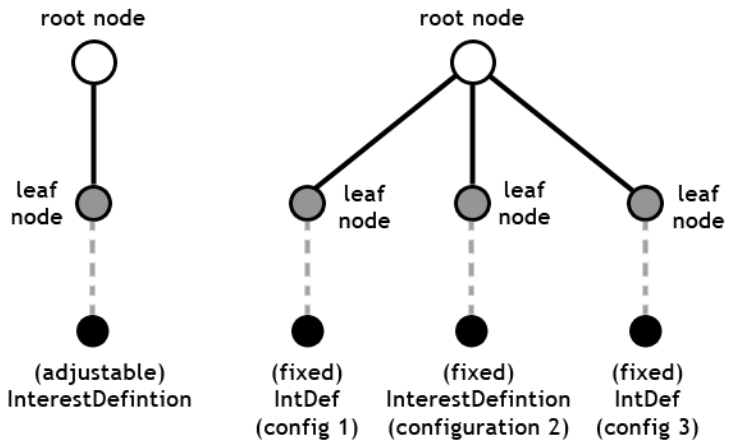

Figure 4 : Possible NIProxy configurations

Note that this tree could have been built quite differently. Most importantly, one could have opted to have multiple InterestDefinitions (and thus child leaf nodes) to represent the various AOIs. In that case, only one InterestDefinition would be active at a given time and switching to a new AOI would be achieved by enabling the correct child leaf node. This situation is visible in figure 4 on the right. For reasons of clarity, this option is not explored further in this paper; however the conceptual workings should be clear to the reader.

When the available bandwidth changes, the root node is notified. This in turn tells the leaf node how much bandwidth it can use. It is up to the implementation of the leaf node to decide which adjustments to the LOD will lead to an optimal bandwidth usage. For this, the leaf node has a list of possible adjustments to the single InterestDefinition, called configurations. For each configuration there is an associated heuristic. This heuristic should provide an indication of how much bandwidth this configuration would use if enabled. The leaf node chooses the configuration with the most appropriate expected bandwidth usage (the highest still under the limit) and tells the Area of Interest system to make the necessary changes to the InterestDefinition, in this way applying the new configuration. The configurations and heuristics for this scenario can be found in table 1.

Best case heuristics		**Worst case heuristics**	
Configuration number	Bandwidth Estimation	Configuration number	Bandwidth Estimation
0	0	0	0
1	530	1	6000
2	2040	2	2040
3	3600	3	300

Table 1 : Heuristics, Configurations and initial Bandwidth Estimation numbers

Note once again that this would be different in case multiple leaf nodes would be used, one for each individual configuration. Under those circumstances, the InterestDefinitions would not have to be adjusted by the Area of Interest system; the NIProxy would just enable the correct leaf node and by extension the correct InterestDefinition.

4.2. First Person Shooter Sniper Scenario - Results

For this scenario, two different tests were run. The first one uses good starting heuristics that provides a realistic indication of how much bandwidth a configuration will use, i.e. a best case scenario. The second test shows the worst case and discusses what happens when the chosen heuristics are completely wrong. The goal of this comparison is to find out whether the system will be able to deal with these conditions and what the likely errors would be. Figure 7.10 shows these two different setups.

Figure 6 shows how the different configurations tie to specific AOI shapes and LODs. Configuration 3 is the most extensive and has high LOD with large circle and wedge shaped areas. Configuration 2 is designed to leave out the inner opponent so the sniper can focus on his current target, i.e. the outer opponent in his wedge area. Configuration 1 drops the outside areas and only sends the position updates of the player himself. This is quite radical but indicates a situation where the bandwidth becomes extremely low and the game will be nearly unplayable, but it is still important to make the game feel a little responsive to the player. Configuration 0 disables even these player updates should the bandwidth drop below the absolute limit of 510 bytes/sec (the amount a single player's position updates use).

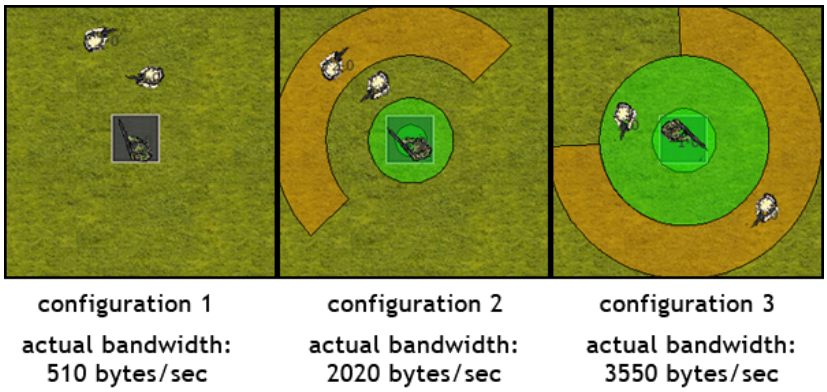

Figure 5 : Configurations for the first person shooter scenario

Figure 7.11 illustrates the different bandwidth limits the simulation goes through. The scenario starts at a very high limit of 10000 bytes/sec, so configuration 3 is chosen. At 20 seconds, the limit drops to 540 bytes/sec. This causes the chosen configuration to switch to number 1 and the AOI is adjusted accordingly, so only the positions of the sniper are sent. At 40 seconds, the limit is once again increased to 10000 bytes/sec and configuration 3 is enforced once more. At 60 seconds, the limit becomes 2300 bytes/sec and configuration 2 is appropriately chosen. After 80 seconds, the limit goes down to the very low level of 400, which is below even the bandwidth usage of configuration 1, so all traffic is effectively shut down. At the 100 second mark, the simulation wraps, starting at the 540 bytes/sec limit and proceeding to 10000 bytes/sec, 2300 bytes/sec etc.

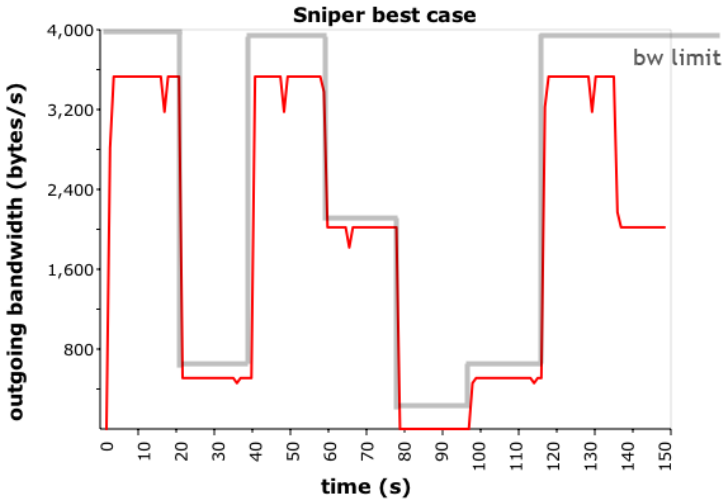

Figure 6 : Results from the first person shooter scenario with optimal heuristics

As can be seen in the graph, due to the accuracy of the heuristics and this approach, bandwidth limits are not exceeded at any time and the most appropriate AOI is chosen at all times. Of course, this is because the simulation was specifically designed to show that, given a good heuristic of the bandwidth usage, the NIProxy bandwidth shaping approach can be effectively used in this virtual environment setup. The next simulation makes it clear that the results are quite different if the heuristics are not realistic.

In this simulation, the bandwidth limits fluctuate in exactly the same way as they did for simulation 1. However, it can be seen that there are some distinct differences in the resulting bandwidth graph shown in figure 7. This is solely due to the use of a different heuristic for the various configurations, as no other adjustments were made to the simulation.

Figure 7 : Results from the first person shooter scenario with worst-chosen heuristics

At first, the limit is 10000 bytes/sec. According to the heuristic, configuration 1 uses the highest amount of bandwidth, so it is selected. After one second of measurements, the heuristic is updated to reflect the actual bandwidth usage (which is actually just 510 bytes/sec). This causes the implementation to switch configurations to the now most appropriate configuration, number 2. This can be seen at the beginning of the graph: there is a very small (1 second wide) bump of 510 bytes/sec before switching to the higher usage of 2020 byte per second. The most important thing to note here is that, even though the bandwidth limit is respected, the available bandwidth is used sub-optimally. The

optimal usage would be for configuration 3 (as we can see in the best case scenario). However, because the initial heuristics for configuration 3 are very low, it is never even considered.

When the limit switches back to 540, configuration 1 is appropriately selected. The reader is reminded that its heuristic has been updated after the measurements in the first second, so that the heuristic is now correct. When the limit switches back to 10000, configuration 2 is once again selected, continuing the sub-optimal bandwidth usage for this limit. The switch to the 2300 bytes/sec limit at approximately 60 seconds also does not result in a configuration switch. At the end of this interval, when the limit switches to 400, an interesting effect is demonstrated. The heuristic for configuration 3 is still 300, which is known to be much too low for its actual usage. Still, the algorithm selects configuration 3 and for a second it is allowed to be used. This makes for the huge bandwidth spike to the highest possible bandwidth usage of 3550 bytes/sec, while the configuration is actually supposed to use only 400 bytes/sec. However, the situation is quickly adjusted after the bandwidth measurements made during this one second. The heuristic for configuration 3 is updated, causing the implementation to choose configuration 0 because all heuristics are now above the limit of 300.

Now that once each of the heuristics has been updated, the simulation will continue to run normally with optimal bandwidth usage (as in the best case scenario).

Note that if a limit below 510 would never have occurred, the wrong heuristic for configuration 3 would never have been detected and the bandwidth usage for higher limits would always be sub-optimal. This means wrong (initial) heuristics can not only cause sudden bandwidth spikes, but can also have severe consequences that lead to continued sub-optimal bandwidth usage. It is interesting to investigate what would happen if an intermediate level would have too high of a heuristic, while the higher and lower levels have a correct heuristic. Under these circumstances, when the limit becomes lower, the simulation will switch from the higher to the lower configuration because it does not know the intermediate configuration has the wrong heuristic and that it is actually the most appropriate option. This can prompt very big swings in consistency and AOI shape, which are of course to be avoided.

This comparison between best and worst case provides some very interesting conclusions, even though the testing scenario itself was very simple. First of all, it is shown that, with the exception of a few spikes, the bandwidth limits are maintained very well. The spikes that occur due to incorrect heuristics only last for a second and are quickly detected and repaired. So even though there might be a few of these anomalies, the algorithm works relatively well compared to what could happen if we would not work with a heuristic or estimation and just try every configuration until an appropriate one for that moment was found.

5. Conclusions and future work

It has been shown that the combination of ALVIC-NG as generic framework for Networked Virtual Environments and the NIProxy as an adaptation layer is able to deliver information in an effective way to individual clients. By using this approach, streams can be individualized for each participant, something that is not possible using a generic NVE framewore, where each client is supposed to accept the same information stream, independent of contextual parameters (i.e. bandwidth, game genre,…). Although only one experiment was described here, the combination has been tested under various circumstances, and its viability has been proven. It has also been shown in further studies that care has to be taken to optimize the processing on the proxy servers, to make sure that they do not become overloaded (slowing down the experience for all players). The implementation of these optimized algorithms and a more thorough investigation into the side-effects of a migration to cloud platforms are currently underway.

Acknowledgements

This work is partially funded by the IWT OMEGA project.

References

[1] ALVIC-NG: state management and immersive communication for massively multiplayer online games and communities. Peter Quax, Bart Cornelissen, Jeroen Dierckx, Gert Vansichem, Wim Lamotte. Springer Multimedia Tools and Applications, vol. 45,no. 1-3,2009
[2] Globally Distributed Content Delivery. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. IEEE Internet Computing, September/October 2002, pp. 50-58.
[3] SecondLife. Linden Research Inc. http://www.secondlife.com
[4] The NIProxy: a Flexible Proxy Server Supporting Client Bandwidth Management and Multimedia Service Provision. Maarten Wijnants and Wim Lamotte. In Proceedings of the 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007). Helsinki, Finland, June 18-21, 2007
[5] Networked Graphics: Building Networked Games and Virtual Environments. Anthony Steed and Manuel Oliveira. Morgan Kaufmann; 1 edition (December 4, 2009).

Biographies

Peter Quax is a post-doc researcher at the Expertise Centre for Digital Media of Hasselt University. He was and is actively involved in a number of international and national research projects and teaches MsC degree courses on multimedia, computer networks and security.

Wouter Vanmontfort is a researcher at the Expertise Centre for Digital Media of Hasselt University. He is currently involved in a national project targeting the deployment of the back-end infrastructure of massively multiplayer on-line games on cloud platforms.

Robin Marx was an MsC student at Hasselt University at the time the results in this paper were generated. He currently heads his own company targeting game and multimedia application development.

Maarten Wijnants is a post-doc researcher at the Expertise Centre for Digital Media of Hasselt University. He was and is actively involved in a number of national research projects and is the main engineer behind the NiProxy architecture discussed in this paper

Wim Lamotte is full professor at the Expertise Centre for Digital Media of Hasselt University. He was and is actively involved in a number of international and national research projects and teaches BsC and MsC degree courses on multimedia and computer networks. He is the lead of the research group on Networked Virtual Environments at EDM.