Revamping Cloud Gaming with Distributed Engines

Revamping Cloud Gaming with Distributed Engines

Luigi De Giovanni, Davide Gadia, Paolo Giaccone, Dario Maggiorini, Claudio E. Palazzi, Laura A. Ripamonti, German Sviridov

***Abstract*—While cloud gaming has brought considerable advantages for its customers, from the point of view of cloud providers, multiple aspects related to infrastructure management still fall short of such kind of service. Indeed, differently from traditional cloud-ready applications, modern game engines are still based on monolithic software architectures. This aspect precludes the applicability of fine-grained resource management and service orchestration schemes, ultimately leading to poor cost-effectiveness. To mitigate these shortcomings, we propose a Cloud-Oriented Distributed Engine for Gaming (CODEG). Thanks to its distributed nature, CODEG is capable of fully exploiting the resource heterogeneity present in cloud data centers, while providing the possibility of spanning its service on multiple network layers up to the edge clouds.**

Index Terms: Cloud gaming, distributed game engines, service provisioning and management.

Introduction

Running a video game requires a specific set of computational resources to execute game-related tasks (e.g., rendering, physics simulation, etc.). In legacy environments, those resources are provided by means of PCs or gaming consoles physically available to the player. Yet, a growing population of players is reluctant to purchase this kind of hardware due to its cost, or being afraid of not taking full advantage of it. Moreover, gaming hardware requires periodic upgrades, thus furthermore limiting the attractiveness of such local solutions.

In recent years, the gaming industry has grown thanks to the advent of online gaming, which generally involves the gaming experience being partially provided with the support of remote servers. Nevertheless, online players still have to rely on local PCs or consoles and face QoE and latency issues [1], [2], [3].

Cloud gaming tries to address the aforementioned limitations by providing a subscription-based service model to play [4]. The cloud provides remote acceleration of some of the heavyweight tasks, thus enabling new games to be enjoyed on otherwise underpowered devices [5]. Such tasks are offloaded to a server located within some private data centers equipped with dedicated hardware. In some cases, such as Google Stadia, Amazon Luna, GeForce Cloud Gaming and Microsoft xCloud, this involves the complete offloading of the Game Engine (GE). This means that the game now runs in a physically dislocated place, leaving to the players' equipment only the task of displaying a pre-rendered game scene.

Cloud systems generally provide resource sharing and service disaggregation across multiple servers in the data centers. Yet, legacy GEs have been traditionally built following a monolithic architecture with multiple elements tightly intertwined. While adapting GE architectures to better exploit the distributed and heterogeneous

resources at the core and the edge of the network is not an easy task, it can bring considerable advantages from the point of view of both operators and players.

Indeed, cloud resources utilization, players' satisfaction, and cost-effectiveness can be improved thanks to the software-defined network infrastructure provided by modern telecommunication network infrastructures. As a motivating example, 5G allows appropriate (i) placement of the GE modules within the network and (ii) allocation of computation and network resources, in order to provide guaranteed performance, especially in terms of maximum latency experienced by the players, thanks to the support of network slicing [6].

Distributed GEs encompass the low latency enabled by edge computing and the advantages in terms of cost and resource elasticity provided by cloud computing. At the same time, distributing GEs across multiple physical devices opens new opportunities for developing hybrid cloud gaming solutions. Indeed, we envision a next-generation of cloud gaming according to which GEs can be distributed across the entirety of the network, from the player's device up to the remote clouds, with the most critical and latency-sensitive parts of the GE placed close to the player. This is coherent with previous work leveraging the edge computing paradigm to reduce network delay and save bandwidth cost [7]. The main idea is to offload computation intensive tasks from the end devices by moving the 2D/3D graphic rendering and the main game logic to the edge node. The cloud is responsible for all the configuration and management tasks without latency constraints. Similarly, the graphical rendering operation can be split between the cloud and client [8].

In this context, we propose CODEG (Cloud-Oriented Distributed Engine for Gaming) as a framework to develop and operate distributed GEs in a software-defined network infrastructure. The goal of CODEG is to fully exploit the heterogeneity of resources present in modern cloud and edge networks and to provide an improved cloud gaming experience while reducing the operational costs for cloud providers. To this end, we will discuss the main architectural requirements and the implications of modern network infrastructures on the design of gaming services.

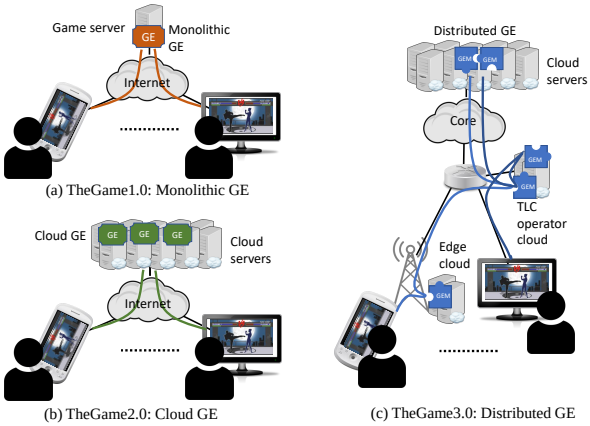
An Overview of Legacy Game Engines

Game Engines (GEs) are software frameworks devoted to the creation of an interactive digital artifact, i.e., a video game. A GE merges various kinds of multimedia assets (e.g., 3D meshes, textures, sounds, and animations) with code describing the interaction between a simulated environment and a player to create an executable program. This executable program is, at its core, an interactive discrete event simulator running the game. To achieve this result, a subset of the GE itself, called *runtime*, is transferred into the game executable to manage resources and schedule events. On the other hand, all other management utilities, i.e., the development toolset, is left behind because not useful while playing.

A GE is usually organized as a software stack rooted in the operating system with an increasing level of abstraction, layer-by-layer, up to a point where game mechanics and interactions are described. Inside a running game, the lowest level of the runtime is acting as an interface to the kernel and the (sometimes proprietary) hardware. This layer is taking care of implementing basic game functionalities such as physics simulation and graphics rendering in an extremely optimized way for the hardware in use. As an example, rendering requires some preliminary processing to be performed on the CPU, but it relies on the Graphics Processing Unit (GPU) for the majority of the operations: the sequence of the different hardware operations on CPU and GPU can become a bottleneck for the overall GE if not correctly managed.

The upper layers of the runtime provide first stubs to undisclosed, proprietary, APIs, then a degree of platform independence, and finally game-specific functionalities, like e.g., Artificial Intelligence (AI) and online multiplayer management.

The traditional architectural model of a GE is the cause of three major shortcomings [9]. First, due to its high performance nature, a game is often a monolithic piece of software, hence the developers must rebuild/relink the whole project (or large part of it) at every change in the codebase. For huge repositories, a global rebuild may become a significant production bottleneck. Second, the vast majority of CPU workload is on the centralized server, while on the client we observe



(a) TheGame1.0: Monolithic GE

(b) TheGame2.0: Cloud GE

(c) TheGame3.0: Distributed GE

Figure 1: Examples of Game Engine (GE) scenarios.

mostly rendering activity. Centralized services are usually hard to scale and perform poorly with a high offered workload. Third, every game exposes some degree of platform dependency. Even if an engine claims to be cross-platform and uses its lowest layer to adapt to vendor-specific hardware, seamless deployment across multiple platforms is not always possible.

Technical Limits of Legacy Game Engines

To show the limits imposed by the architecture of current-generation GEs, and introduce our proposal, we will illustrate three evolving scenarios.

Scenario 1 (Monolithic GE). A video game company, "*TheCompany*", is producing and providing an online multiplayer game: "*TheGame*". As in Fig. 1(a), TheGame 1.0 has been developed following a standard monolithic approach: the server-side is built as a single piece of software and the service is made available to the public via a high-performance machine and a high capacity network connection. This approach has been used by popular games in late 90s such as Quake and, more recently, by Minecraft (Java Edition). TheGame 1.0 proved to be a very successful game: on day one, many players connected to enjoy it. Unfortunately, the system under strains immediately shows its weaknesses: (i) uneven response latencies, (ii) connection dropping, and (iii) poor performances due to system overload. Indeed, each player connects to TheGame through a different network path and using many different access technologies and, thus, uneven network delays are experienced; players with lower delay will typically have some advantage on the other players, affecting the overall game fairness. Furthermore, connection dropping or excessive delays can be caused by an increased number of concurrent network connections exceeding the physical server capacity. While it is technically possible to boost the hardware, this is unlikely to meet TheCompany budget limitations. Moreover, after the initial hype, the workload will significantly drop, and many resources will be left unused.

Scenario 2 (Cloud GE). To overcome scalability issues, TheCompany adopts a new strategy to deploy TheGame 2.0: a cloud infrastructure, as shown in Fig. 1(b). Players can now connect to the service through multiple servers, usually geographically closer to the players, greatly reducing the delay. The software is still deployed as a monolithic installation, but multiple copies of the game servers running on virtualized machines can be available. The cloud management system is able to elastically adapt the resources to the demand by running servers proportionally to the number of connected players. Many modern games with large audience use this approach. We can mention Second Life, Candy Crush, and Clash of Clans; all using Amazon Web Services (AWS). See [10] for a list of case studies on AWS.

Even with the adoption of a cloud approach, TheCompany is still facing several issues. Firstly, despite having multiple servers, all players are interacting in the same environment and, thus, all the game instances must refer to the same global state. Synchronizing in real-time the game state across all servers is very complex and thus it is typically preferable to refer to a shared state available centrally in the cloud infrastructure. This is the case of the virtual realms for legacy Massively Multiplayer Online Games (MMOG), such as *World of Warcraft*, in which the bottleneck due to the shared state has been limiting the maximum number of players allowed in each realm. Secondly, the monolithic nature of the performance-oriented software typically imposes strict constraints on the hardware and software choice, boosting performances but severely limiting the scalability of the approach, like in the case of Microsoft xCloud, where Windows is required as hosts' operating system, or GeForce

Cloud Gaming, running only on NVidia proprietary hardware [3], [4].

Towards Scenario 3 (Disaggregated GE). To solve all the issues mentioned so far, when developing TheGame 3.0, TheCompany must completely change its paradigm. Given the limitations described in the previous sections, we can safely suggest that the architecture of current GEs is not adequate to provide the flexibility and scalability required by game developers and designers of the next generations. An alternative proposal for a more efficient architecture is to decompose a GE in several dynamic and independent software modules interacting with each other via a microkernel-like message bus, thus defining a disaggregated GE. This approach can offload all computational effort to the game infrastructure, including rendering. This could contribute to improve the scalability of game streaming architectures such as Stadia and Luna.

The SMASH engine described in [11] is an execution environment following such approach. SMASH provides three basic functionalities: a soft real-time scheduler, a dynamic game modules manager, and a messaging system between modules. The game modules are independent entities providing gaming functionalities. A module may implement a general-purpose engine service, such as graphic rendering or physics simulation, or be extremely-game specific. The possibility to swap in and out modules at runtime allows developers to effortlessly modify and extend games with a plug-in approach, down to a very fine granularity. Each software module uses a message bus to call service functions provided by other modules.

The Price of Centralization

We consider 256 real topologies of network providers available from the Internet Topology Zoo [12] and we study the latency experienced by a player if the GE was centralized in a single cloud, as in Scenario 2. We assume (optimistically) that all the players belongs to the same network provider and the data center is optimally located to minimize the latency by any other node. Table 1 shows the worst case game latency, i.e., the Round Trip Time (RTT) experienced from any node to the closest cloud, taking into account only the propagation delays of communication links. The latency could be much larger than 30 ms, which have been observed as the maximum acceptable delay for highly interactive games [13]. Replicating the game on 8 data centers would decrease the propagation delays by a factor 5, leaving about 23 ms on average to accommodate further latency associated to distributed implementation (e.g., queuing delays, function calls, computational overheads, replication overheads). This motivates the need of carefully distributing the GE across the network.

Table 1: Performance in 256 topologies of network providers

Num. GE clouds	Worst case game latency [ms]		
	min	ave	max
1	1.9	32.2	161
8	0.48	6.4	24

Distributed Game Engines in the Cloud

In the context of cloud computing, the modular and structured nature of disaggregated GEs represents a cornerstone for the development of fully distributed ones. Modern cloud infrastructures are composed of a set of Compute Nodes (CNs) which can be either physical or virtual servers. Furthermore, thanks to the advanced network virtualization techniques offered by modern telecommunication networks, it is possible to define logical networks with specific Quality-of-Service (QoS) levels, thus providing minimum bandwidth and maximum delay guarantees for the game-related traffic. Leveraging on this capabilities, enabling *Scenario 3*, TheCompany can then develop TheGame 3.0 by abstracting a disaggregated GE as a mesh of microservices running into distinct containers (e.g., Docker ones) on different CNs, as shown in Fig. 1(c). We claim that this approach, coupled with the traditional benefits of distributed systems, can enable truly distributed, scalable, and fault-resilient GEs with high load adaptability and little-to-no downtime.

We baptized our proposal for a distributed GE running on a cloud infrastructure CODEG: Cloud-Oriented Distributed Engine for Gaming. Differently from SMASH [11], where the focus was to modularize a GE and to propose a plug-in system to improve the development process, CODEG is designed to provide a *network-wide abstraction* of an operating system for GEs. This

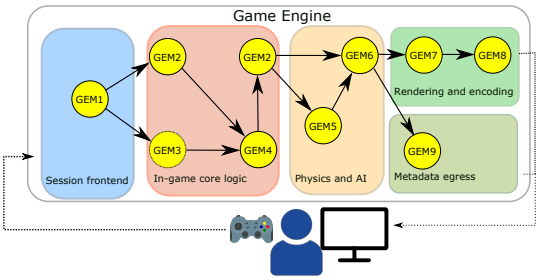


Figure 2: Game session workflow in CODEG.

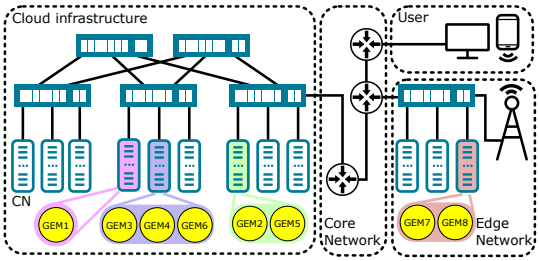


Figure 3: Mapping of GEMs on the underlying hardware in CODEG

abstraction must be capable of exploiting multiple CNs offered by a modern cloud infrastructure. While not being an easy task, abstracting each disaggregated game module through a mesh of microservices living in different parts of the cloud represents a major architectural improvement over legacy solutions for the design of cloud-ready distributed GEs. Achieving such a result requires considerable effort on multiple levels: (i) the definition of distributed GE modules and their location inside the CNs, (ii) the mitigation of the overhead for synchronizing shared states across many CNs, and (iii) the QoS support to control end-to-end network latency. In the following, we analyze the challenges along with the benefits that CODEG brings to game developers, designers, and players.

CODEG Workflow

To reach its scalability and elasticity goals, CODEG partitions each GE into a set of disaggregated modules, namely GEMs (Game Engine Modules). A GEM represents a basic element of a given GE and it is activated whenever its functionality is required. As in conventional GEs, individual GEMs must be chained together to implement more complex functionalities. For a given game, the GEMs, alongside their interconnection, form the GE abstracted by CODEG. During the execution, GEMs continuously exchange information among each other, in order to implement the GE processing workflow.

Fig. 2 depicts an example of a CODEG instance, i.e., a set of GEMs required to run a single game. Such an instance can in turn be associated with one or more game sessions or players. The possibility of running multiple game sessions per CODEG instance largely depends on the game and session type. For example, in the case of multiplayer game sessions, some GEMs provide a conventional multiplayer game backend, thus requiring only one GEM instance holding the same global state for many game sessions. In other scenarios, the GEM instance is stateless and, thus, can be reused for different players and different games at the same time. Finally, some other GEMs are unique for each player, thus requiring a new instance of a GEM for each session.

While the GEMs responsible for running the main in-game logic can be shared across multiple sessions, each game session is assigned a unique pair of *ingress* and *egress GEMs*. The ingress GEMs are responsible for managing per-session information and permit to discriminate across different sessions. Whenever a player input is received at the ingress GEM, the information contained in the request is propagated through the entirety of the GE, and session-specific output is generated at the egress GEM. As the egress GEM is disaggregated from the rest of the GE, it enables the possibility of supporting different operation modes for each session. Notably, for fully cloud-based gaming solutions, an egress GEM can be comprised of a remote rendering pipeline capable of video streaming, while for hybrid or legacy solutions, such output would be composed of metadata in order to perform local rendering of the in-game world at the player's side.

GEM Placement Strategy

The main peculiarity of CODEG is that it enables the possibility for each GEM to be located in an arbitrary position inside the cloud infrastructure, as shown in Fig. 3. Yet, while an arbitrary set of CNs can be selected for such a task, they have to satisfy requirements in terms of underlying

resources availability and, most importantly, have to meet application-level requirements in terms of response latency, as perceived by the player. This calls for a suitable *placement* strategy for each GEM, taking into account the processing and network resources allocated for the game session.

Processing resources Each GEM has a particular set of hardware requirements that must be satisfied by the underlying hardware of the CN. Such is the case of rendering-related GEMs which require GPU rendering capabilities inside the CN, or physics simulation GEMs which may require a given amount of RAM and CPU. Thus, the placement strategy must take into account the availability of suitable and enough computation resources.

Network resources As in legacy GEs running on a local operating system, the CODEG workflow requires dedicated communications between the GEMs. This implies the existence of a logical network across different CNs, e.g., as enabled by the network slicing capability offered by 5G networks. Tailored QoS requirements (e.g., maximum delay and minimum bandwidth) can be either associated to all or groups of players (for a coarse but simpler network management) or to a single player (for an accurate but more complex management). The requirements in terms of bandwidth are dictated by the nature of each GEM, and consequently, by the type and temporal pattern of exchanged data.

Costs and performance Public cloud service providers offer different types of CNs depending on the actual needs of the customer. While being different in terms of provided resources, these CNs differ also in terms of leasing costs, reflecting different computation and network infrastructures, as shown in Fig. 3. The low-cost conventional cloud infrastructures can be used for more resource-demanding but latency-insensitive tasks, while the edge cloud for more latency-sensitive GEMs, at the expense of an increased cost.

An optimal GEM placement strategy minimizes the total cost for the game provider, while satisfying a maximum response delay for all the players, taking into account the actual load due to the active game sessions and the current available computation and network resources. When no feasible solution is found, either some new players' sessions are blocked or additional resources are reserved.

CODEG Load Adaptability

A major advantage of CODEG with respect to fully-centralized solutions is its load adaptability. In general, the players' activity pattern is not stationary and typically shows day-night, weekly and seasonal effects. The number of concurrent game sessions varies over time, and thus the overall load on the GE is very dynamic. Furthermore, despite the preliminary resource allocation in terms of computation, storage and network resources, unexpected events (as network failures or performance interference in the cloud) can occur, triggering new resource allocations. Notably, resource overutilization will lead to application-level performance degradation, thus poor quality of experience for players, whereas resource underutilization will lead to higher operational costs. Thus, it is of paramount importance to dynamically allocate and adapt the resources to cope with the variable gaming demand and the variable resource conditions.

In traditional approaches exploiting monolithic GEs, scalability is achieved by instantiating multiple copies of the whole GE [14] and adopting some load balancing scheme, leading eventually to a large resource overhead associated with each instance of the GE. Instead, CODEG adopts a GEM orchestrator that provides fine-grained migration and replication of individual GEMs depending on the measured load. In the case of resource overutilization, the orchestrator will provide new candidate CNs into which migrating the affected GEMs. If no viable candidates are found, additional copies of such GEMs will be instantiated by the orchestrator, thus providing application-layer load-balancing across different instances of the same GEM. On the contrary, in the case of resource underutilization, some GEMs could be either exploited to serve requests from other CODEG instances or could be migrated to a location with a lower operational cost. Thanks to the natural disaggregation and containerization of GEMs, such functionalities can be enabled by out-of-the-box solutions already present in most

of the commercial orchestrators.

Response Latency-aware Design

While minimizing the operational costs and satisfying the underlying hardware constraints, in CODEG the GEM placement must be designed to provide guaranteed response latency for the players. The response latency depends on the processing time in each GEM and on the overall network delay, which comprises the delay experienced by the player's commands to reach the ingress GEM, the delay to transfer the data between GEMs when running in different CNs, and the delay from the egress GEM to the player. In the case of shared states between GEMs, additional delay could be experienced due to the adopted replication protocol.

A key aspect in the placement is given by the degree of GEM co-location resulting from the placement phase. Indeed, a GE fully co-located inside the same CN would degenerate into a traditional GE and minimize the communication overhead. However, such a strategy falls short in terms of load adaptability, as previously discussed. Distributing the GE across multiple CNs inevitably adds the communication latency between the GEMs to the perceived response delay, and this delay degradation is the main price to pay in CODEG as opposed to legacy monolithic and cloud GEs. Communication latencies depend on the allocated bandwidth and on the amount of exchanged data between the GEMs. Thus, characterizing the dependency graph between the different GEMs in terms of data is crucial to properly predict the communication latency for a given placement of GEMs within the network. For this reason, possible solutions to minimize the response latency may employ placement strategies aimed at colocating in the same CN groups of highly connected GEMs and/or data-intensive GEMs.

QoS support in the network allows to control the communication latency, but this requires characterizing the traffic leaving each GEM, in terms of the amount of data and its burstyness. The logical network can be defined at different granularity levels, depending on the capabilities offered by the network control plane: in a 5G network, we envision a single network slice shared by multiple game sessions, in which the actual bandwidth is allocated based on the worst-case burstiness and bandwidth of the data transferred between GEMs.

In summary, the placement strategy should minimize the overall operational cost while satisfying the maximum response time for the players and taking into account resources costs and availability. Reducing delays may require the use of resources at the edge of the network but at higher costs than conventional clouds. Yet, thanks to the disaggregated nature of CODEG, one may employ hybrid solutions by spanning the GEMs across multiple network layers. In this way, more latency-sensitive GEMs can be placed close to the player while maintaining the less latency-sensitive but resource-demanding GEMs at the network core, ultimately leading to a better trade-off between the player's experience and operational costs. As suggested in Fig. 3, rendering and encoding GEMs can be placed at the edge of the network, thus offering the possibility of transmitting synthetic metadata over a congested core network while transmitting the game video stream only in the proximity of the player.

Future Research Directions and Challenges

In the near future, we foresee a number of research fields converging into this topic because of the innovative large-scale service model on both the player and provider sides. From an engineering standpoint, network-aware distributed real-time cloud applications will raise the bar for networking support by requesting end-to-end cross-infrastructure with QoS guarantees, as well as optimization tools to define efficient and effective placement strategies. To this end, we expect that GEMs should be properly profiled and that novel static and dynamic placement policies, aware of delay and bandwidth overheads, need to be devised. From a player-experience perspective, new ways to evaluate players' experience will be required in order to explore the inter-dependencies between game elements and infrastructure performance. Game content creators will be able to leverage on a virtually infinite resource pool and, on the other hand, the extreme scalability in terms of the number of concurrent players will call for novel and extended players' experience approaches. On an economical stand-

point, we might also assist to a radical change in the value chain due to new pricing and business models. E.g., the players might be charged for the resources used while playing regardless of the actual game. This approach may end up blurring the distinction between game publishers and service/cloud/network providers, opening innovative service models for online games. Last but not least, our approach may also support Virtual Reality platforms (e.g., Facebook Meta) where users connect to and interact with each other.

Conclusion

We propose CODEG, a Cloud-Oriented Distributed Engine for Gaming, according to which a game is implemented by combining multiple modules (GEMs) and placing them across the available network infrastructure, exploiting heterogeneous computation, storage, and network resources. The main design requirement for allocating the resources is to minimize the operational costs while satisfying a maximum response delay constraint. We discuss how CODEG can be integrated into modern telecommunication networks by distributing the GEMs across the core resources and the edge ones. In conclusion, we argue that the proposed solution is capable of reaching a sweet spot between player's quality of experience and operational costs.

Acknowledgments

This work is partially funded by the Department of Mathematics of the University of Padua through the BIRD191227 project.

REFERENCES

1. J. Saldana and M. Suznjevic, "QoE and latency issues in networked games," *Handbook of Digital Games and Entertainment Technologies, eSpringer*, 2015.
2. A. Bujari, M. Massaro, and C. E. Palazzi, "Vegas over Access Point: Making room for thin client game systems in a wireless home," *IEEE Transactions on Circuits and Systems for Video Technology*, 2015.
3. M. Suznjevic, L. Slivar, and L. Skorin-Kapov, "Analysis and QoE evaluation of cloud gaming service adaptation under different network conditions: The case of NVIDIA GeForce NOW," in *QoMEX 2016*, 2016.
4. A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, and D. Giordano, "A network analysis on cloud gaming: Stadia, GeForce Now and PSNow," *Network*, 2021.
5. W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. M. Leung, and H. C.-H., "A survey on cloud gaming: Future of computer games," *IEEE Access*, 2016.
6. X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, 2017.
7. X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, "Improving cloud gaming experience through mobile edge computing," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 178–183, 2019.
8. J. Bulman and P. Garraghan, "A cloud gaming framework for dynamic graphical rendering towards achieving distributed game engines," in *USENIX HotCloud*, Jul. 2020.
9. D. Maggiorini, L. A. Ripamonti, and G. Cappellini, "About game engines and their future," in *EAI GOODTECHS*, 2015.
10. Amazon Web Services, "Gaming Customer Success Stories," https://aws.amazon.com/gaming/gaming-customer-references/.
11. D. Maggiorini, L. A. Ripamonti, E. Zanon, A. Bujari, and C. E. Palazzi, "SMASH: A distributed game engine architecture," in *IEEE ISCC*, 2016.
12. "The Internet Topology Zoo," http://www.topology-zoo.org.
13. G. Sviridov, C. Beliard, A. Bianco, P. Giaccone, and D. Rossi, "Removing human players from the loop: AI-assisted assessment of gaming QoE," in *IEEE INFOCOM Workshop*, 2020, pp. 1160–1165.
14. H. Liu and M. Bowman, "Scale virtual worlds through dynamic load balancing," in *IEEE/ACM DS-RT*, 2010.

Luigi De Giovanni is Associate Professor of Operations Research at Università degli Studi di Padova, Italy.

Davide Gadia is Assistant Professor of Computer Science at Università degli Studi di Milano, Italy.

Paolo Giaccone is Associate Professor of Telecommunication Networks at Politecnico di Torino, Italy.

Dario Maggiorini is Associate Professor of Computer Science at Università degli Studi di Milano, Italy.

Claudio E. Palazzi is Associate Professor of Computer Science at Università degli Studi di Padova, Italy.

Laura Anna Ripamonti is Assistant Professor of Computer Science at Università degli Studi di Milano,

Italy.

German Sviridov received his PhD in Electrical, Electronics and Communications Engineering from Politecnico di Torino, Italy.