Pragmatic Internet Egress Routing for Improving Player Latency in Interactive Multiplayer Games

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in Computer Science

at

The University of Waikato

by

Oladimeji Fayomi

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2024

Abstract

Interactive multiplayer games rely on the network to ensure a smooth and responsive experience. When the latency between a game server and a client is beyond a threshold, the client is unable to play the game properly and often abandons the game. We posit that this unfortunate disconnection is avoidable in certain cases. Because Border Gateway Protocol (BGP) is agnostic to latency, the paths used by clients may be suboptimal latency-wise. Providing performance-aware routing on the Internet in general is a hard problem, and proposals to change BGP or the core of the Internet struggled to gain traction. However, Multiplayer games provide a workable problem: it may be possible to attain *feasible* (i.e., sufficiently low) latency for players without major changes to the Internet core.

This thesis presents a *pragmatic* latency-aware routing control approach which tackles interactive multiplayer games and other applications that need to operate under some specific latency threshold but are not bandwidth-hungry. The proposed approach – Overwatch, leverages BGP Egress Peer Engineering (EPE) and capitalises on the well-connected nature of game provider networks to offer multiple paths to reach their clients. By selecting a feasible path for each client, Overwatch can maximise the number of clients/players in the game. One of the key advantages of Overwatch is its deployability, which is enabled by its explicit routing feature using commodity hardware, which is interoperable with existing devices and does not require collaboration with other networks. Overwatch controls traffic along feasible paths according to an operator-defined algorithm. The current implementation is based on a Segment Routing over IPv6 (SRv6) dataplane. This implementation was used to evaluate Overwatch in a scenario with 300 clients. Prior to that, we ran a set of Internet measurements involving relevant game servers in Europe to calibrate the experiments. The evaluation highlights Overwatch's benefit in selecting feasible paths for the players when such paths exist, thus keeping players online.

Acknowledgements

In the spirit of a Yoruba proverb that resonates deeply with me: *"Oju merin lo n'pe omo waye, sugbon igba oju ni wo d'agba."* I reflect on my journey. This translates to, "A child may be born from two people, but it takes a village to raise the child." Indeed, I am a testament to this communal upbringing.

I would like to begin by acknowledging and expressing my profound gratitude to my supervisors, Dr. Marinho Barcellos, Dr. Richard Nelson, and Dr. Matthew Luckie. Their patient guidance and invaluable mentorship over the years have been instrumental in the completion of this work.

My deepest appreciation goes to my wife, Gracie Adeyinka-Fayomi, for her unwavering support and understanding throughout this journey. Her encouragement has been my bedrock. To my mother, and my parents-in-law, your endless support and prayers have been my guiding light, for which I am forever grateful. My sisters, Folakemi and Temi, your support and constant encouragement have been a source of strength.

I am grateful to friends who have become like family: Omoniyi Alimi, Mr. and Mrs. Salami, Lola Bamgbose, and Ben Amadi. Your support has been invaluable. To my former bosses and colleagues at Catalyst Cloud, now dear friends – Bruno Lago, Chris Weeks, and James Dempsey – thank you for your support and for paving the way for this adventure to begin.

I extend my appreciation to the WAND Research group members: Brad Cowie, Richard Sanger, Florin Zaicu and Chris Lorier. A special mention to Dr. Magnos Martinello for his invaluable assistance. There are many more who have supported me in various ways throughout this journey, and while I cannot name everyone, your contributions have been deeply appreciated.

Lastly, I would like to express my gratitude to the University of Waikato for the doctoral scholarship and the departmental funding that supported this work.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AS Autonomous System

ASN Autonomous System Number

BGP Border Gateway Protocol

CBE Container-Based Emulation

CDF Cumulative Distribution Function

devRTT Deviation of RTT

EPE Egress Peer Engineering

EWMA Exponentially Weighted Moving Average

IGP Interior Gateway Protocol

IP Internet Protocol

ISP Internet Service Provider

MED Multi-Exit Discriminator

MPLS Multiprotocol Label Switching

NLRI Network Layer Reachability Information

NOS Network Operating System

PAR Performance-Aware Routing

PoP Point of Presence

QoE Quality of Experience

QoS Quality of Service

RAM Random Access Memory

RIB Routing Information Base

RPC Remote Procedure Call

RTT Round-Trip Time

SCION Scalability, Control, and Isolation On Next-generation Networks

SDN Software-Defined Networking

SR Segment Routing

SRH Segment Routing Header

SRv6 Segment Routing over IPv6

TE Traffic Engineering

WAN Wide-Area Network

Chapter 1

Introduction

1.1 Problem Statement

Context. Online gaming is a huge market, worth approximately US$26.14 billion in 2023 and is projected to reach US$34.11 billion by 2028 [136]. It is a subset of video games in which the players' machines are connected and played through the Internet or any other computer network [2]. According to [35], online game genres can be categorised based on two properties of player actions: the "precision" required to complete an action and the "deadline" by which the action must be completed. Online game genres such as First-Person Shooters (FPS), sports, racing and fighting demand high precision and tight deadlines [36]. Player actions in these genres focus on completing tasks that require hand-eye coordination and motor skills [2]. On the other hand, genres such as Real-Time Strategy (RTS), Role-Playing Games (RPG) and adventure games have low precision and loose deadlines [36]. The gameplay, which defines how players interact with the game in these genres [121], focuses on planning and skilful thinking rather than the completion of physical tasks [75].

Online gaming application logic. Regardless of their genre, online games are discrete real-time simulation applications [62]. In these applications, the game is represented as a continuously running loop during which everything happening is calculated, loaded, rendered and displayed [103]. These applications

have time constraints to perform these tasks at a high frequency because of the need to display the output in real-time [161].

Online gaming communication models. Online games rely on two major communication models: peer-to-peer and client-server architectures [2]. In peer-to-peer architectures, no entity participating in the game has more control over the game than others. In client-server architectures, a designated entity mediates and manages the game, thus giving it control over others [157]. Client-server architecture is the prevalent architecture for online games [116, 157, 35]. In traditional online gaming client-server architecture, game state computation and update tasks occur on the server entity [103]. In contrast, in cloud gaming, a new form of online games, game state computation, update and rendering tasks happen on the server entity and the game is streamed as video to the game client [154, 89]. This allows the game client to be lightweight, playing out the streamed video like a video player and sending player input to the server [154, 155].

Interactive multiplayer games. This work focuses on client-server interactive multiplayer online games. They enable multiple players to connect, collaborate or compete in a shared game environment in real time. High-profile examples include Counter-Strike: Global Offensive (CS: GO) [38], Call of Duty [80], and League of Legends [59]. The operation of interactive multiplayer games from an Internet standpoint can be abstractly described as follows. An authoritative server handles the game logic and state processing. Meanwhile, clients run an interactive application and communicate with the server, which collects and redistributes the game state [35].

Performance requirements. These interactive multiplayer games rely on the network and its optimal performance for a smooth and responsive experience. They are especially sensitive to "ping time": when the latency[1] between a game server and a client exceeds some specific threshold, the client is unable to play the game properly [106, 94, 107] and often abandons it.

[1]The terms latency and Round-Trip Time (RTT) are used interchangably.

To function correctly, a client needs to be able to exchange packets with the server under a *feasible* (i.e., sufficiently small) latency. While having the "lowest" latency possible between the client and server enables a more responsive experience for some multiplayer online games, minimising it is irrelevant in most other multiplayer online games as the server typically works step-wise and does not distinguish between feasible latencies. The clients are scattered around a server, and in practice, the physical distance between them is constrained by latency. While latency is an issue, the client-server communication in interactive multiplayer game applications considered in this work has a *low* bitrate requirement [41, 27]. (Note that the other category, cloud gaming, also requires a high bitrate [25, 40, 1, 103]). Besides latency and bandwidth, the performance of interactive multiplayer games is also sensitive to jitter and packet loss [106]. Most multiplayer online game applications respond to jitter as they would respond to latency exceeding a threshold [60]. Meanwhile, the effects of packet loss are mitigated by updating the game state frequently or packet repair techniques [36]. As such, feasible latency remains the primary requirement for interactive multiplayer game performance.

Internet routing is latency-oblivious. As we will demonstrate later, game provider networks are typically well-provisioned in terms of connectivity. However, the best path with respect to latency for each client may differ from the default path selected from available transit providers and peers [100]. This difference is often due to the routing policies of the provider or peer offering the default path. In this context, the challenge originates from the performance obliviousness of Border Gateway Protocol (BGP), the Internet's de facto inter-domain routing protocol. BGP may make poor choices when selecting a path because it prioritises factors like the number of Autonomous System (AS)-level hops or the age of paths rather than crucial performance metrics such as latency [117, 92, 134]. Also, BGP inherently limits the utilisation and propagation of multiple viable paths that may be available due to its implicit withdrawal and path-hiding behaviours [149, 159]. Additionally, BGP lacks

the agility to modify its path choices in response to performance fluctuations without causing disruptions due to its slow convergence and chatty nature [73, 87].

Problem. In some instances, the Internet paths used for client-server communication in interactive multiplayer game applications may be suboptimal latency-wise due to the performance obliviousness of BGP. This suboptimal path choice may cause the latency between the server and clients to exceed the threshold and thus prevent these clients from connecting and staying in the game. Additionally, modest increases in latency may degrade the performance and players' experience in FPS games [94].

Prior work. The goal of guaranteeing latency (and bandwidth) for applications is longstanding, with an extensive body of research and work that has explored various solutions to fulfil this goal. Nevertheless, many interesting and promising proposals, such as [163, 64], have not gained traction. This failure of prior proposals to achieve widespread adoption can be attributed to several challenges, such as the difficulties of incremental deployment, the lack of immediate incentives, the lack of backward compatibility with current inter-domain routing technologies and the need for collaboration across multiple networks [129]. According to [118], for any solution to be genuinely impactful, it must be grounded in pragmatism, working within real-world constraints, even if it only provides modest gains.

1.2 Proposed Solution

Insight. A recent study [105] highlighted that paths other than those deemed best by BGP can yield considerable improvements in latency. Also, an older study indicates that alternative paths offer reduced latencies in 30 – 80% of cases compared to BGP's choices [122]. Furthermore, ample alternate routes are available due to the diversity of paths on the Internet and robust connectivity between networks – attributed to the strategic placement of Points

of Presence (PoPs) by content and cloud providers across diverse global locations [10]. This increased connectivity and diversity of paths on the Internet thus present a valuable opportunity to exploit the lower latencies on alternate paths and to provide feasible latency for interactive multiplayer game applications.

Hypothesis. This thesis posits that it is possible to design a *pragmatic system* that maximises the number of players in a multiplayer game by adjusting the server-to-client paths when the selected BGP paths fail to meet the latency requirements.

Proposed solution. The proposed approach utilises BGP Egress Peer Engineering (EPE) [53] to steer flows from game servers to players by considering latency. As such, this *pragmatic* latency-aware routing control system steers server-to-client traffic along better paths when the initially selected BGP path fails to meet the latency requirements. The design capitalises on the fact that game provider networks are well connected, and, as previously stated, significant latency improvements can be obtained by using a path different from the one selected by BGP. This approach achieves this goal while providing incremental deployability without requiring collaboration with other networks.

1.3 Contributions

This thesis offers four main contributions. First, it introduces Overwatch, a BGP EPE [53] routing control system designed for pragmatic Performance-Aware Routing (PAR) in networks. The work explores the limitations that make it non-trivial for BGP to provide inter-domain latency-aware routing for applications. It discusses the requirements that prevent the widespread adoption of prior proposals to provide latency-aware routing. Informed by these insights, this thesis presents a design approach for Overwatch that uses EPE to provide PAR. This conceptual design is then realised in implementing an Overwatch prototype using SRv6 on a Linux-based BGP EPE-enabled node, which is subsequently evaluated.

Second, we develop a lightweight algorithm for Overwatch. This algorithm incorporates routing information, topology details, and latency measurements to maximise the number of clients with latency under the threshold. Specifically, it is tailored for controlling the server-to-client traffic between a game server and its players. The algorithm can be modified/extended according to specific application requirements.

Third, we analyse publicly available information, including routing tables from various vantage points and Internet topology, to validate the proposed approach's potential benefits and shape an evaluation scenario. This analysis aimed to confirm the availability of multiple transit providers for game service provider networks. Additionally, we measured latency between game service provider networks (*game networks*, for short) and networks with clients located within ≈100ms latency from the game servers. These measurements confirm the viability of the proposed approach in delivering latency-aware routing for players in many networks accessing these servers and inform the subsequent evaluation.

Fourth, the Overwatch prototype was run in an IPMininet [18] emulated environment, whose topologies and end-to-end latencies were configured according to the Internet measurements performed. This assessment offered valuable insights into the effectiveness of Overwatch and validated its performance. The results of our assessment demonstrated that Overwatch can indeed maximise the number of players in a game, by ensuring that the paths with feasible latency are chosen whenever one or more exist. In scenarios where changing network conditions caused excessive latencies for players connected to a server, our evaluation showed that Overwatch successfully kept all of the players connected to the server. These results confirm our hypothesis that designing a system that maximises the players in a multiplayer game is possible when latency is excessive for the players on the paths selected by BGP.

Furthermore, the Overwatch prototype's source code and the emulation environment's details, including topologies and configuration, have been made

available.

1.4 Thesis Structure

Chapter 2 lays the groundwork for this thesis, introducing the necessary background information and problem specification. It explores the key concepts and themes discussed, mainly focusing on the performance requirements and need for PAR for multiplayer online game applications and BGP's limitations in meeting this need. This chapter also analyses Internet routing tables and related measurements to emphasise conditions and scenarios where Overwatch can be beneficial.

Chapter 3 reviews other possible approaches that could be used to maximise the number of players in a game, such as changes to the Internet architecture, application-layer mechanisms, improvements to game network infrastructure, and pragmatic PAR routing approaches.

Chapter 4 outlines the design choices and principles that fulfil the pragmatic latency-aware routing requirements. The chapter also provides an overview of the architecture of the routing control system that incorporates the design choices and principles.

Chapter 5 presents the implementation of the architecture established in Chapter 4. This chapter details how existing hardware capabilities, open-source applications and stable and standardised software with commodity hardware were used to implement Overwatch's architecture components. It also explains how the implemented components interact to provide latency-aware routing for multiplayer online games.

Chapter 6 evaluates Overwatch's implementation to assess the benefits of the latency-aware routing control system. It describes the use of a container-based emulation approach for the evaluation, explaining the choice of this method. The chapter covers the metrics used to evaluate Overwatch, the emulation environment's experimental setup, and the interpretation of the

results. It concludes with a discussion of the limitations of the evaluation.

Chapter 7 explores the application of Overwatch's routing control system to other networks. It discusses how additional functionalities could be added to Overwatch to provide PAR for other applications besides multiplayer online games. Lastly, the chapter expands on how the flow steering subsystem in Overwatch can be implemented in network environments with equipment from different vendors.

Chapter 8 concludes the thesis, summarising the research findings. It identifies areas for improvement in the latency-aware routing control system and suggests directions for future research to address the limitations identified.

Chapter 2

Problem Specification

This chapter introduces the concepts required to understand the thesis. Section 2.1 provides an overview of the interactive multiplayer online games that are the focus of this work, their specific performance requirements and how meeting these requirements impacts player numbers and game experience. Section 2.2 describes why Border Gateway Protocol (BGP) cannot meet the performance requirements that interactive multiplayer online game applications demand from the network. Section 2.3 discusses the conditions required on the Internet to circumvent the performance obliviousness of BGP and enable Performance-Aware Routing (PAR) for interactive multiplayer online game applications. Section 2.3 also presents an analysis of Internet topology and BGP routing data to demonstrate the prevalence of path diversity on the Internet, especially for game provider networks. This section also highlights how leveraging alternative paths can improve performance for these interactive multiplayer online game applications.

2.1 Interactive Multiplayer Game Application Requirements

Application description. Typically, most online games require these three elements:

1. The "game world" is a virtual and artificial environment that players experience and engage in via the output methods provided by the computer, such as screens and speakers [83].

2. The "gameplay", which describes the challenges and activities in the game world and the actions that players should perform to solve those challenges or do the activities [121, 90].

3. The "avatar" is a representation of the player in the game world, the player controls it in the game world to perform actions [16].

In this sense, interactive multiplayer online games (multiplayer games for short) are applications that allow multiple geographically dispersed players to compete or collaborate in a shared game world simultaneously over the Internet or a network. In effect, the Internet or network functions as the medium through which those players can interact in the game. Although some multiplayer game genres do not fit this description, this description applies to the relevant genres for this work.

Classification of multiplayer online games. According to [75], multiplayer games can be classified into five broad genres based on game elements such as the gameplay, player actions and their impact on the game world. The genres are:

1. Mini-game/Puzzle, a genre in which the gameplay involves logic and conceptual challenges [2]. Players typically do not have to be playing at the same time. Minesweeper [104] and Jigsaw Explorer [131] are examples of such multiplayer games in this genre.

2. The adventure genre describes multiplayer games in which the players' avatars perform exploration in the game and solve puzzles in the game world [2]. An example of a game in the genre is Minecraft [138]. They are similar in many ways to games in the Role-play genre, which will be described next.

3. The Role-play genre describes multiplayer games in which players perform tactical, logistical and exploration challenges such as collecting loot in the game world and trading for better weapons [2]. Examples are Star Wars: The Old Republic [57] and World of Warcraft [43].

4. Resource genre includes sub-genres such as Real-Time Strategy (RTS), Turn-Based Strategy (TBS) and Construction and Management Simulation (CMS) games [75]. In this genre, the gameplay elements are cognitive, and solving challenges requires planning instead of fine motor skills. Examples are Age of Mythology [137] and Command and Conquer: Generals [58].

5. The Action genre includes sub-genres such as First-Person Shooters (FPS), sports, racing and fighting [75]. In games such as Call of Duty [80], FIFA 23 [133], Need for Speed: The Run [11] and League of Legends [59], the experience is perceptually driven, and they place significant demands on the player's physical skills [8].

According to [35], the genres described above can be further categorised based on the "precision" required to complete an action successfully in the game world and the "deadline" by which the action must be completed. Games in the action genre demand high precision and tight deadlines, while all other genres mostly have lower precision and looser deadlines [36]. The relevance of classifying game genres based on precision and deadline to this work will be apparent later in this section.

Communication Model. As previously mentioned, online games rely on two major communication models: peer-to-peer and client-server architectures [2]. However, the client-server architecture is the predominant architecture for multiplayer games [116, 157, 35] because it typically allows the game provider to host and manage the server which controls the game state in commercial multiplayer games. This server control makes it easy for game providers to maintain game state consistency and prevent cheating [147, 82]. Also, the

client-server architecture is more straightforward to implement because there is no need for peer discovery and distributed storage management required in peer-to-peer architecture [85]. Multiplayer cloud games also rely on client-server architecture. However, in their case, all computation, processing, and rendering to update the game world is performed on the server, and the game is streamed like video to the client [154, 89]. Meanwhile, in the client-server architecture used in conventional multiplayer games, the game client still performs some rendering and processing of the game world [98, 127]. This approach used by multiplayer cloud games has benefits such as decreased hardware and software requirements for clients and the ability to access games anytime, anywhere, independent of the device used [32, 28]. However, due to the specific performance requirements of multiplayer cloud games, as will be discussed later in this section, this work focuses on something other than multiplayer cloud games.

Application logic. Multiplayer game applications such as Counter-Strike: Global Offensive (CS: GO) [38] involve a client and server exchanging messages at regular intervals known as "ticks" [103, 132]. These ticks determine how frequently the server updates the game state and transmits these updates to the client. The client transmits player inputs (actions) to the server, and the server, in turn, processes these actions to update and send a new game state back to the client at every tick. The client then processes and renders the new game state and collects the player input for the game state. These ticks also impact the "acceptable latency threshold" between the client and server, which will be discussed later. Some recent multiplayer game applications allow the processing of messages between ticks, thereby providing faster updates to clients with latencies much lower than the acceptable latency threshold. However, this model generated some controversy in the game communities, with some players complaining about strange behaviour and unfairness [37]. We focus on the most common model, i.e. without updates between ticks, and assume that minimising the latency further beyond the acceptable threshold

will not provide gains.

Requirements. Generally, excessive latency, bandwidth, loss and jitter negatively impact multiplayer gameplay [103, 106]. However, different studies have shown the impact of bandwidth to be marginal on gameplay in typical client-server multiplayer games because of the small size of the messages exchanged between the server and clients [41, 27, 100]. According to [14], packet loss as high as 5% has no measurable effect on player performance in multiplayer games. Also, the effect of loss in multiplayer games can be mitigated by frequent game state updates and packet repair techniques [36]. In previous studies, small amounts of jitter do not affect performance and gameplay experience [41, 124]. In contrast to occasional packet losses and jitter, extreme jitter impacts multiplayer game performance [71]. Nevertheless, most multiplayer online game applications respond to extreme jitter as they respond to latency exceeding a threshold [60].

Latency, the delay between a player's action and the corresponding outcome in the game world, negatively impacts player performance and overall gaming experience [156, 93, 42]. As a result, many high-profile multiplayer games, such as Counter-Strike: Global Offensive (CS: GO) [38], Call of Duty [80], and League of Legends [59], have specific latency thresholds [41]. Exceeding these game-specific thresholds can drastically hinder the user experience, making the game virtually unplayable [106, 94, 107]. Furthermore, due to inherent latency constraints, the physical distance between the server and clients is typically confined to a geographic area, with a geographically distributed set of servers covering different regions. As discussed later, these constraints may limit the number of clients connecting to a server. Therefore, for these applications, the priority is the swift exchange of packets between the clients and server, ensuring that latency remains below a specified "feasible latency" threshold.

Latency does not affect all multiplayer games equally. According to [36], the impact of latency on games depends on the precision and deadline required to perform players' actions. RTS and RPS multiplayer games have low precision

and loose deadlines because the nature of their gameplay actions favours cognitive skills and strategy over real-time interaction. Studies have observed that latencies ranging from hundreds of milliseconds to several seconds do not affect the performance and experience of RTS and RPS games [77, 34, 128, 56]. On the other hand, interactive multiplayer games such as Counter-Strike: Global Offensive (CS: GO) [38], Call of Duty [80], and League of Legends [59] demand high precision and strict deadlines, and are susceptible to latency [36]. This work aims to meet the latency criteria demanded for interactive multiplayer game applications requiring high precision and tight deadlines.

Maximising players in a server. This work aims to maximise the number of players connecting to a game server. This is because many multiplayer games allow players to choose the server they will connect to, and this choice matters to players because of the game maps, configuration and competitions that may be available on a particular server. However, game server selection tools only consider the perspective of a single client, and a server might offer different performance and experience to different geographically dispersed players [63]. Additionally, the authors in [96] observed that players in the same city with different Internet Service Providers (ISPs) might experience different latency to the same game server because of their provider's routing choices. As a result, fewer players, instead of more, can join a game server and have a good experience, leaving players unsatisfied due to the cascading effect of latency [78, 162]. We believe it may also leave the game servers underutilised. This challenge has given rise to a niche industry of "game acceleration", committed to mitigating latency concerns through Gamers Private Network (GPN) solutions [152, 44] and Wide-Area Networks (WANs) optimised for gaming traffic [139]. However, these GPNs and WAN have limitations, as will be discussed in Chapter 3. With this in mind, this work aims to maximise the number of players connecting to a game server by ensuring "feasible latencies" for these players independent of the paths selected by BGP.

2.2 Performance Obliviousness of BGP

Lack of performance-awareness in Internet routing. Internet routing can often be sub-optimal, mainly due to the lack of insight into the performance of a route. An inter-domain route traverses several distinct Autonomous Systems (ASes), each characterised by its own set of policies, Quality of Service (QoS) benchmarks and capacities. Consequently, individual ASes along an inter-domain route might face issues that adversely impact the performance of traffic traversing such a route. On the other hand, BGP – the primary inter-domain routing protocol responsible for path selection – is performance-oblivious and remains static in its routing decisions, even when network conditions fluctuate. In the event of performance degradations or shifts in network conditions on a route, manual interventions to adapt network policies and adjust configurations across multiple devices are often necessary. These interventions happen because BGP's route processing is restrictive and lacks provision for real-time adaptation. However, given the intricacies of the protocol and its range of adjustable parameters, applying network changes becomes a challenging and high-stakes endeavour, especially when timely reactions to performance issues are essential [49].

BGP limitations. Central to the challenge is that the primary role of BGP is the scalable dissemination and exchange of Network Layer Reachability Information (NLRI) among ASes. The success of BGP in fulfilling this role has enabled the growth of the Internet to accommodate a diverse range of applications and networks. Nevertheless, this growth is placing significant strain on the capabilities of BGP and stressing several of its inherent limitations [79, 159]. The pertinent limitations of BGP, in the context of this work, include (i) a lack of multipath routing, (ii) constrained Traffic Engineering (TE) capabilities, and (iii) an absence of QoS support.

2.2.1 Lack of Multipath Routing

Multipath routing requirements. Multipath routing generally describes a category of techniques and approaches that make concurrent use of multiple paths to transmit traffic between a specific source and destination. Multipath routing offers several benefits, including the ability to load-balance traffic across multiple paths, improved fault tolerance, and optimised network resource utilisation. With these benefits, networks can achieve performance objectives such as reliable communication, higher throughput, reduced latency, and congestion control. Fundamentally, multipath routing requires three core capabilities: (i) the ability to identify or determine multiple paths, (ii) the capacity to forward packets via multiple paths, and (iii) the ability to distribute traffic across multiple paths, considering various factors like load-balancing, congestion control, and latency reduction [130].

BGP lacks multipath ability. In contrast, BGP, in its standard configuration, opts for and advertises a singular *best route* to a specific destination, even when multiple routes exist. When a new route for the same destination is advertised, it replaces the current route, a phenomenon known as *implicit withdrawal* [149, 159]. While an extension [146] to BGP introduced a capability that allows the advertisement of multiple routes for the same prefix, it addresses only one aspect of the multipath routing prerequisites. BGP inherently lacks the capability to send packets through multiple paths and effectively distribute traffic across them based on specific needs. Further complicating the matter is that for multiple paths to be advertised via BGP, the peers must support this capability and the specific dataplane optimisations required to utilise these additional paths. Additionally, the extension bloats the routing table, as additional routes have to be stored for each destination.

2.2.2 Limited TE Capabilities

TE is typically implemented to enhance network reliability and improve the performance of the network and its carried traffic. Achieving these TE objectives

demands practical traffic steering ability. However, BGP offers minimal tools for implementing TE for some reasons that will be examined shortly.

Lack of multipath affects TE. Firstly, as just noted in Section 2.2.1, BGP does not support the advertisement of multiple routes to the same destination. Also, it cannot distribute traffic among various routes. Consequently, it is hard or even impossible to spread traffic across multiple paths to achieve TE goals such as improved throughput and congestion management.

Internet architecture and policies do not encourage TE. Secondly, the decentralised architecture of the Internet means there is no global coordination and cooperation of routing policies among networks. BGP allows each AS on the Internet to decide which reachability information it shares with its neighbours using a set of static, AS-specific policies. As a result, any given AS might apply its local policies, routing its outbound traffic as it deems fit. These local policies applied in an AS could potentially conflict with the routing preferences of a downstream AS. Therefore, it is challenging to precisely control and distribute incoming traffic to an AS across different paths using BGP. This is because inter-domain traffic flows in a direction opposite to route advertisements, as an AS can only send traffic to a prefix that it has its route.

Tuning BGP parameters does not provide TE. Several techniques rely on manipulating BGP attributes – including selective advertisement, AS-path prepending, Multi-Exit Discriminator (MED), and communities – to influence inbound traffic to an AS [24]. However, these methods may not always work, as the BGP attributes that they rely on may not be recognised by external ASes. Even when the necessary BGP attributes are recognised, these strategies aim to influence the routing decisions of external ASes by tweaking these attributes. Neighbouring ASes are not obligated to honour the policies suggested by the manipulation of these attributes, as they are not mandatory attributes. Therefore, ASes can ignore these tweaks without violating the BGP specification.

Due to BGP allowing each AS significant autonomy in setting routing

preferences through statically defined policies, managing inbound traffic via BGP is a manual and iterative task. This process depends heavily on precise tuning and configuration of BGP attributes and parameters, with a successful outcome far from guaranteed.

2.2.3 Lack of QoS Support

BGP must incorporate network quality metrics into its routing decisions to provide QoS assurances for various applications. However, BGP lacks the mechanisms to dynamically evaluate the network's performance or traffic and respond to changes in network conditions. Its primary design intent was to distribute reachability information scalably, which means BGP does not inherently communicate critical network metrics like capacity, bandwidth, or cost that may be necessary to infer QoS. BGP does not factor these metrics in its routing decisions either. Instead, it prioritises factors like the length of AS-level hops and the age of the paths [92, 117, 134].

An extension proposed to BGP [153] could empower BGP routers to advertise QoS metrics and select routes based on available bandwidth. However, to be effective, this would require modifications to BGP and collaborative efforts between ASes.

2.2.4 Summary

The lack of multipath routing, the limited TE capabilities, and the lack of QoS support in BGP collectively render the protocol performance oblivious. However, replacing or changing BGP is clearly impractical. Consequently, this thesis focuses on an incrementally deployable approach that can realistically provide inter-domain PAR, even if it means accepting modest improvements. The next section describes a network scenario in which the proposed PAR, a latency-aware routing approach, would be beneficial.

2.3 Motivating Use Case and Analysis

Latence-aware routing prerequisites. Maintaining the latency between a game server and a client below a specific latency threshold is crucial for ensuring a smooth and responsive experience for multiplayer game applications, as discussed in Section 2.1. However, inter-domain routing achieved with BGP may not always follow the best possible path in terms of throughput and latency [122, 134], as just discussed. As such, attaining feasible latencies for clients when default BGP paths exceed the acceptable latency threshold hinges on two prerequisites: (i) diverse routing options and (ii) routes with different latencies.

When multiple routes to a destination have different latencies, they can be used to provide latency-aware routing for latency-sensitive applications such as interactive multiplayer games. The topology scenario in which this could happen is outlined in this section to validate the pragmatic latency-aware routing approach to be proposed and confirm its potential gains. Then, the section describes the preliminary Internet measurements and analysis performed to assess the prevalence of multiple routes on the Internet and the variability in their latencies.

Network scenario. The scenario in which a latency-aware routing approach is to be employed is described as follows. There is an application (a game) in which clients need to communicate with a server under some predefined latency threshold. The maximum tolerable latency pragmatically limits the number of clients and their physical distance to the server. For an interactive multiplayer game application, it is typically under 100ms [41, 14, 110]. The network in which the game server resides has multiple paths. Typically, packets transmitted from the game server to clients are routed through one of these paths, the choice of which is made by BGP. Figure 2.1 illustrates a simplified example of the topology for the scenario. We assume that the Game Server AS will have multiple transit providers and peer networks (A–D). The figure shows paths A, B, C and D crossing egress points (A–D) respectively through

ASes A, B, C and D converging to a common point, AS X, and then reaching AS Game Client 1. We investigated the potential for steering clients to lower latency paths in this scenario.

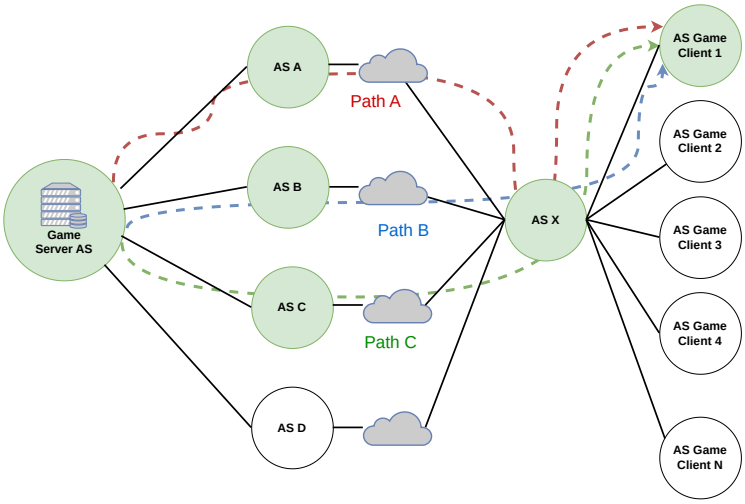


Figure 2.1: Network topology illustrating diverse BGP paths (A,B,C) connecting game server AS to client networks via upstream providers and transit AS X.

2.3.1 Diversity of Paths

Previous studies have established that path diversity exists for most prefixes within large Tier 1 transit providers [31, 140]. Moreover, content and cloud providers have also strategically established Points of Presence (PoPs) in multiple global locations to ensure robust connectivity and enhance path diversity [10].

Path diversity in networks. While it is common for path diversity to exist within large Tier 1 transit providers due to their robust connectivity, there are few large Tier 1 networks compared to other networks on the Internet. These Tier 1 networks can also capitalise on this diversity without violating rules and practices that guarantee inter-domain routing stability and safety. However, this scenario only holds for a few networks on the Internet. This is because of the valley-free rule, as described by Gao-Rexford [61], which states that a customer would not typically transit traffic to a remote destination on behalf of its provider. Due to this rule, non-Tier 1 ASes tend to conserve their network resources, avoiding the transit of traffic for non-customer neighbours.

Path diversity in non-Tier 1 networks. Our primary objective is to demonstrate that many networks possessing at least two upstream transit providers could benefit from a wise upstream selection for specific flows. We analysed networks on the Internet and classified them into three categories based on a snapshot of the CAIDA AS relationships [20] and inferred-AS-to-Organisation mapping [22] datasets. The three distinct categories are:

1. Tier 1 networks that only have customers and peers without providers.
2. Tier 2 and 3 networks that have customers, peers, and at least one provider.
3. Stub networks that exclusively rely on providers and do not have customers or peers.

After filtering out inactive ASes (not announcing any prefixes), 75,999 ASes were left. Among Tier 2 and 3 networks, the use of multiple providers to reach remote destinations is evident, as shown in Figure 2.2: 77% have two or more upstreams, and 10% boast six or more upstreams. In the case of stub networks, these percentages drop to 54% and 1.5%, respectively, as shown in Figure 2.3.

Availability of multiple paths to game networks. Specifically considering the motivating scenario, 11 popular game networks were selected and examined within a specific geographic area (Europe) to assess their use of multiple transit providers. The Autonomous System Numbers (ASNs) of the game networks were identified through information available on their websites and different looking glass services, which were validated with the CAIDA AS rank [19] dataset. Using BGPStream [109], an analysis was performed on a 12-hour snapshot (From 00:00 to 12:00 on October 24, 2023) of the BGP routing tables obtained from collectors of Oregon's Routeviews project [26] and RIPE's Routing Information Service (RIS) [120], all of which are located in Europe. The following was determined:

1. The number of transit providers (an underestimation of the number of different paths) serving these game networks, based on AS paths between vantage ASes[1] in Europe and game networks.

2. Prefixes advertised by these game networks.

3. Vantage ASes that had routes to the prefixes advertised by these game networks, based on routes these vantage ASes advertise to the collectors.

4. Vantage ASes that could reach these game networks through multiple paths (excluding AS paths via peers).

The values are shown in Table 2.1. Of the 11 game networks considered, 8 announce their prefixes via 3+ transit providers. There is a substantial difference between the *least* and the *most* connected game networks: the former could reach at least 7% of the vantage AS networks via multiple paths, while the number for the latter is 98%. Only transit providers for the networks examined were considered to avoid including paths that could potentially violate the valley-free rule, as previously described.

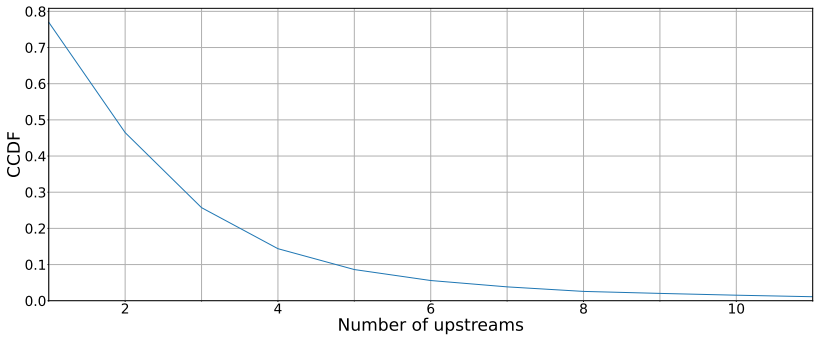


Figure 2.2: Distribution of upstreams in Tier-2 and Tier-3 networks.

2.3.2 Latency Variability in Paths

A recent study has demonstrated that significant improvements can be obtained using paths other than the BGP best paths [105]. Additionally, previous

[1] ASes that establish BGP sessions with route collectors to share their routing tables and updates.

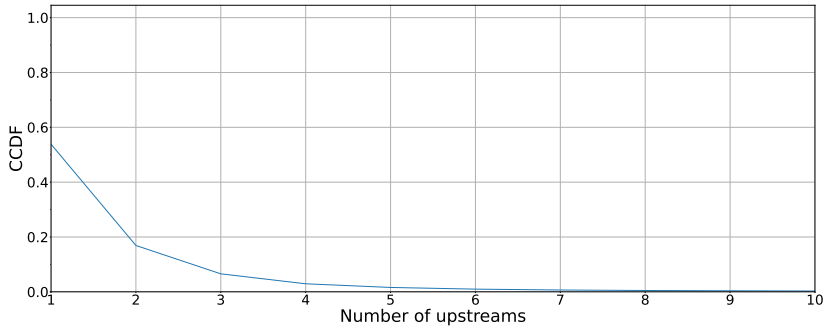


Figure 2.3: Distribution of upstreams in stub networks.

Game Providers	Total Transits	Transits In EU	# of Prefixes	Vantage ASs	Vantage ASs w/ Multiple Paths via transit providers
EPIC Games	1	1	2	237	6.75%
Bandai Namco	2	1	2	247	31.2%
Sega Games	2	2	2	219	7.3%
Nintendo Games	3	3	9	243	98.3%
EA Games	3	3	10	266	90.2%
Ubisoft	6	5	42	233	74.7%
Blizzard Games	12	11	197	292	94.5%
Sony	14	12	117	271	90.8%
RIOT Games	18	16	31	278	94.2%
Take-Two	21	19	63	298	96.6%
Valve	27	25	77	302	97.7%

Table 2.1: Overview of transit connectivity and path diversity in game service provider networks.

research has shown that alternative paths have lower latencies than those chosen by BGP in 30-80% of cases [122].

Measurement objectives. In the defined scenario with multiplayer game applications – where a group of clients need to communicate with a server beneath a particular latency threshold – a recent study highlighted the stark variance in gaming latencies between different ISPs serving gamers in the same city. This disparity was attributed to the distinct peering and path selection preferences of the ISPs [96]. To validate these observations further, we conducted latency measurements between "eyeball" networks (where clients reside) and game servers within a restricted geographical region. The objectives are twofold: First, to ascertain a considerable number of networks within a

latency range that could benefit from performance-aware path selection, and second, to profile the latency to inform the latency-aware routing experiments.

Measurement platform. For the measurements, we utilised probes of the RIPE Atlas platform [135] to ping the IP addresses of three prominent game service providers: Blizzard [17], Ubisoft [141], and Valve [142]. The measurements were performed in Europe due to its dense dispersion of Atlas probes and its hosting of popular game servers, notably in Amsterdam and Frankfurt, which replied to pings or traceroutes.

Validation of measurement targets. To ensure the accuracy of the server locations, the information supplied by the game service providers was cross-referenced with data from MaxMind [99] and IPLocation.net [81] IP geolocation services. The preliminary round of active measurements involved a set of 6,304 probes located in European Union (EU) nations and Switzerland; the set was progressively reduced. During this preliminary round, every probe dispatched a sequence of three ping (ICMP) packets three times, with each sequence separated by 8 hours (spanning 24 hours). Using a 60ms latency threshold value, we excluded probes with median Round-Trip Times (RTTs) to each game server that were either exceptionally low (all three median $\leq$ 40ms) or exceedingly high (all three medians $\geq$ 100ms) since clients in these networks could not benefit from the latency-aware routing solution. This reduced the set to 2,401 probes. Each probe in this set was configured to send a train of five ICMP packets to each of the three game servers at six-hour intervals over 48 hours. After applying additional filtering, there were 1,121 probes across 366 networks. When assessing the game servers individually, the probe numbers were adjusted to 913, 966, and 817, respectively (a probe may be in the desired range for more than one server).

Figure 2.4 presents the RTT measurements from the probes to the three game servers: Blizzard, Ubisoft, and Valve. The measurements are organised by median latency values, with each probe showing minimum and maximum RTT values. For the Blizzard and Ubisoft game servers (Figures 2.4a and 2.4b),

approximately 71% of probes demonstrated stable latency characteristics, exhibiting less than 10ms difference between their minimum and maximum RTT values, with maximum RTTs remaining below 60ms. In contrast, the Valve game server (Figure 2.4c) showed greater latency variability, with only about 50% of probes maintaining this level of stability. These findings suggest two distinct scenarios:

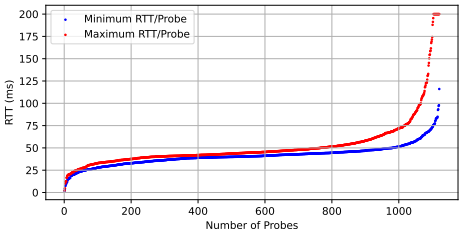
1. Blizzard and Ubisoft servers: While the majority (71%) of probes experience stable latency on their default paths, the networks of the remaining 29% could benefit from latency-aware routing to maintain RTTs below the 60ms threshold.

2. Valve server: Approximately half of the probes experience latency variation on their default paths, indicating a more substantial opportunity for latency-aware routing solutions in these networks.

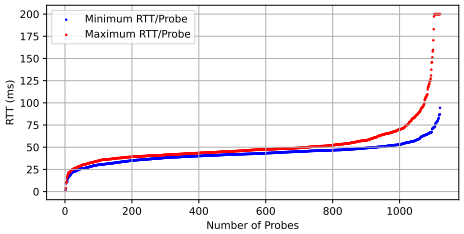
These results demonstrate varying degrees of path stability across three game providers' networks and highlight the potential benefits of implementing latency-aware routing solutions, particularly for networks experiencing significant RTT variations.

Keep probes 2+ AS-hops from the server. Probes (like clients) directly connected to game servers might not benefit from this solution because of the lack of path diversity. In this step, traceroute measurements were performed from the set of identified probes, retaining only those with an AS path that included two or more distinct ASNs between the game server AS and the AS of the probes. To map IP addresses from the traceroutes to their respective ASes, the following datasets were employed: CAIDA routeviews prefix2as [23], RIR extended delegations [3, 6, 9, 88, 119], CAIDA AS relationship [21], and PeeringDB [111]. Each IP address hop in the trace measurements was translated to its corresponding ASN, yielding an AS path. From the probe traceroute, AS paths were generated to the game servers. Any measurements where fewer than three ASNs could be identified were disregarded. A valid

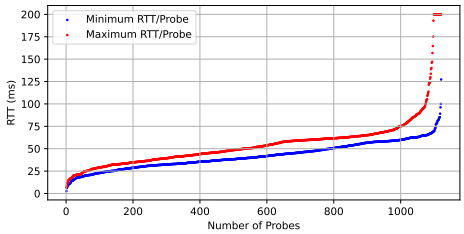
AS path needed at least two intermediate ASNs for the analysis. A path shorter than three might suggest either non-responsive hops or a probe network directly connected to the game server networks.



(a) Blizzard game server.



(b) Ubisoft game server.



(c) Valve game server.

Figure 2.4: Minimum and maximum latency between probes and game servers, with probes sorted by median latency.

Discard offline probes. Some of the probes were offline or unable to complete the measurements, so we failed to map 96 IP addresses to their ASes. However, these IP addresses accounted for less than 4% of the total 3,112 IP addresses observed in the trace measurements. After excluding unresponsive probes and invalid measurements, for the three game servers, 171, 168, and 221 probes 2+ AS-hops away were identified, respectively, located in 112, 73 and 100 ASes, respectively. The traceroute measurements obtained from these probes to the three game servers were analysed, as will be subsequently discussed, to gain insights into the latency variation between different AS paths leading to the game server networks, whenever possible. A latency profile was created from this analysis, which served as a reference for guiding the evaluations.

Creating graphs to represent the topology. In order to analyse

the traceroute measurements, two directed graphs[2] were created from all the valid traceroute measurements and the generated AS paths from the probes (identified earlier) to the three game servers. In the graph created from the generated AS paths, each node represents an AS in the path, and the edges are unidirectional, originating from the probe's ASN and terminating at the game server's ASN. Each node includes additional information about the measurements it appeared in and the number of directly connected nodes. This graph facilitated the ad hoc identification of the set of ASes directly connected to the game server networks. Subsequently, the transit providers for each game server network were identified from this set of ASes using the CAIDA AS relationship dataset [21]. The identified transit providers for these game servers were then used to explore possible AS paths (that do not violate the valley-free rule) between the probes and the game server networks. These paths could be compared using the second graph or subjected to further measurements to understand the latency variation possible between the observed paths to the game servers.

The second graph was constructed from the raw, valid trace measurements. In this graph, each node represented an observed IP address successfully mapped to an AS in the measurements. Similar to the first graph, the edges were undirectional, starting from the probe's IP and terminating at the game server's IP. Each node also contained information about the measurements it appeared in and the number of nodes connected to it. Furthermore, the edges in this graph also included information regarding the *differential* RTT [55] between the connected nodes. This *differential* RTT is calculated by subtracting the RTT from the probe to a hop from the RTT from the probe to the subsequent hop in the measurement. Although the cumulative *differential* RTTs may not precisely reflect the delay along a path due to routing asymmetry on the Internet, they still offer a reasonable indication of the path's characteristics. These values quantify the observed latency variations among different paths

[2]The AS path graph and IP-level RTT graph visualisations are available at [46] and [48].

from an AS to a game server network, as observed in the trace measurements.

Analysing the graphs. We analysed these graphs using an implementation of the Dijkstra shortest path algorithm and depth-first search algorithms[70], focusing on the AS-level connectivity to the game servers. Figure 2.5 presents this analysis as normalised distributions, with all metrics expressed as percentages of the total 366 probe networks used in the measurements. The results revealed that:

1. Two-Hop Connectivity: A substantial fraction of probe networks are positioned at least two AS hops away from the game servers, representing 31% for Blizzard, 20% for Ubisoft, and 27% for Valve.

2. Provider Reachability: A smaller subset of these probe networks can reach the game servers through their transit providers: Blizzard (8%), Ubisoft (3%), and Valve (9%).

3. Path Diversity: The percentage of networks with multiple observed paths to the game servers is notably low: Blizzard (¡2%), Ubisoft (3%), and Valve (5%). This limited path diversity observation aligns with the inherent limitation of traceroute measurements, which primarily capture the selected forwarding path.

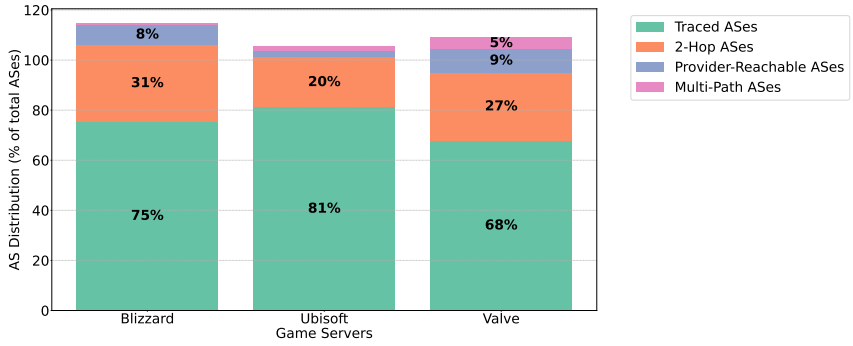


Figure 2.5: Probe ASes used in the trace measurements to identify probes that are two AS hops from game servers and have multiple paths to the game servers.

Latency profiling. Finally, to analyse potential latency improvements through path diversity, we employed *path-stitched ping measurements* to profile

the latency of potential alternative paths identified through the graph analysis. This approach utilises Atlas probes or IP addresses within networks along the paths as anchor points. The *path-stitched ping measurements* were inspired by *Hoplets* [115], a path measurement and analysis approach that combines traceroute measurements to detect poor RTT performance. The accuracy of the results was validated by comparing the approximate RTT measures obtained through these measurements with the cumulative *differential* RTT derived from the traceroute samples.

Due to the limitation of traceroute measurements only capturing selected forwarding paths, detailed RTT analysis was possible for only eight ASes that exhibited path diversity to the game servers (as illustrated in Figure 2.5). The RTT distributions of paths to each game server were analysed using a consolidated approach. For each of the eight ASes that demonstrated path diversity, we identified their best path (lowest median RTT) and worst path (highest median RTT). All the RTT measurements from these paths were then aggregated to understand the overall performance difference between the best and worst paths across all ASes to each game server. Figure 2.6 presents these consolidated RTT distributions as Cumulative Distribution Functions (CDFs) for each game server:

1. Blizzard (Figure 2.6a):
    - Combined RTTs from all best paths show minimal improvement over worst paths.
    - Only 1.6% reduction in median RTT(0.7ms).
    - Suggests consistently similar path performance between the best and worst paths to the game server.
2. Ubisoft (Figure 2.6b):
    - Aggregated best-path RTTs show moderate improvement.
    - 12.9% reduction in median RTT(6.0ms).

- Indicates consistent potential for latency improvement across the ASes.

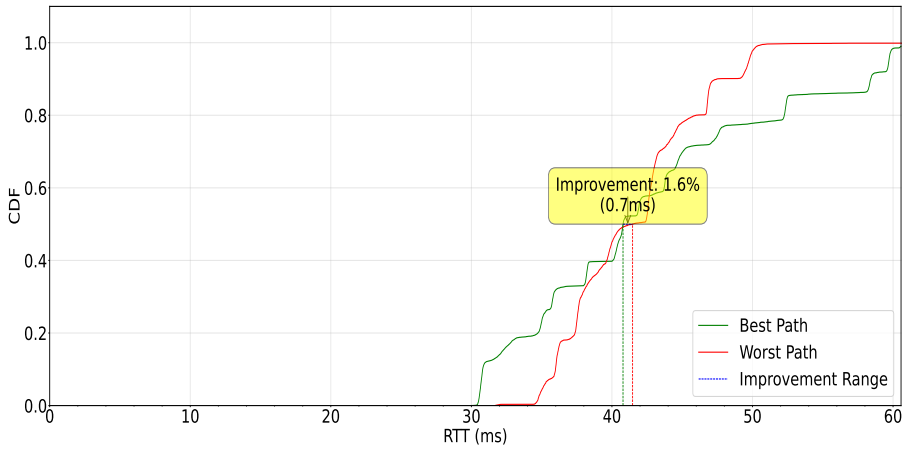
3. Valve (Figure 2.6c):

   - Greatest separation between aggregated best and worst path RTTs.
   - 36.0% reduction in median RTT (18.7ms).
   - Shows significant potential for latency improvement through path selection.

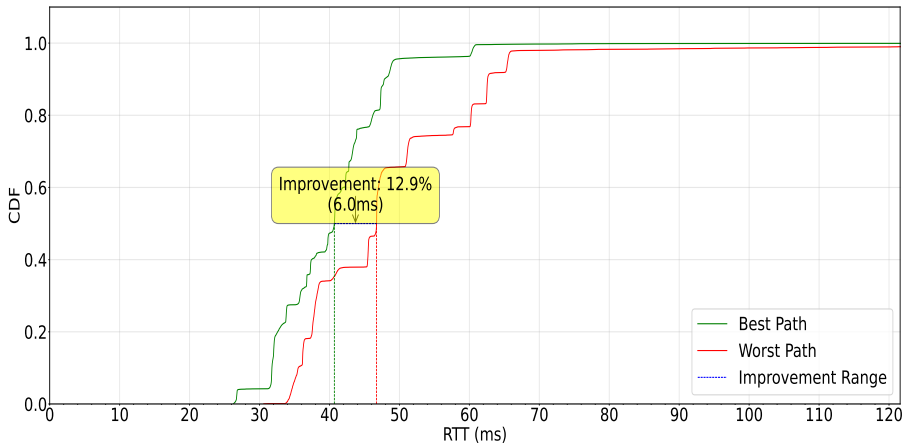
This consolidated analysis reveals that while Blizzard's path shows minimal RTT variation, Ubisoft and Valve servers could benefit from latency-aware routing due to better-performing alternative paths across multiple ASes.

2.4 Summary

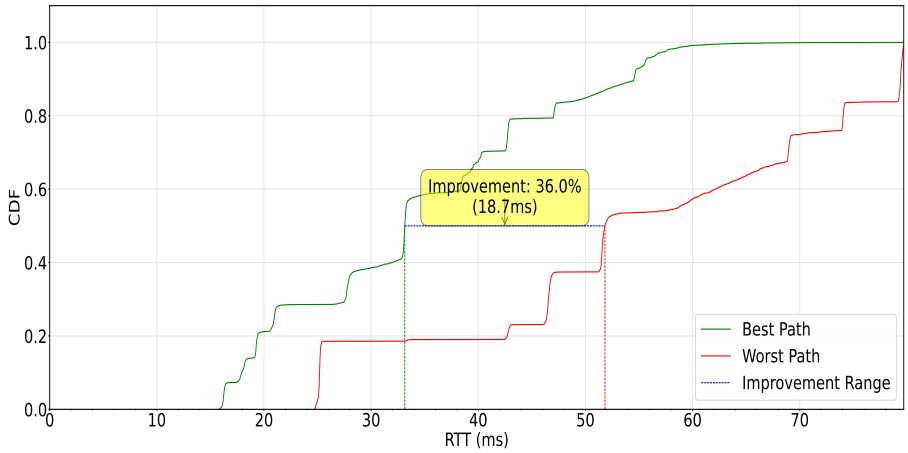
This chapter described the multiplayer games that are the focus of this work, their specific performance requirements, and how these requirements impact the players who can connect to a game server and their experience. It also discussed the inherent limitations of BGP that make it unable to meet the performance demands of multiplayer games. We defined a scenario for multiplayer games in which clients need to communicate with a server under a specified latency threshold to validate our intuition and proposed solution. In this scenario, the proposed latency-aware routing solution aims to ensure feasible latencies for more players where possible, thereby keeping them in the game. To substantiate this scenario and demonstrate the feasibility and benefits of the proposed solution, we analysed path diversity using publicly available Internet topology and BGP routing data. The analysis aimed to illustrate that a significant number of networks, especially game providers, with at least two transit provider connections, stand to benefit from strategically selecting paths for specific flows.



(a) Blizzard game server.



(b) Ubisoft game server.



(c) Valve game server.

Figure 2.6: Improvement in RTT between worst and best paths from probe ASes to the game servers.

Latency measurements were also conducted between eyeball networks and game servers within a restricted geographical region. The aim was to ascertain

the presence of many networks within the latency bracket that a latency-aware routing solution can help and to profile this latency to inform subsequent experiments to evaluate the solution. The findings show that many networks could benefit from a wise choice of paths for specific flows. Furthermore, many of these networks fall within the latency range suitable for a latency-aware routing solution. The following chapter examines the limitations of existing inter-domain PAR approaches.

Chapter 3

Related Work

The problem stated in the previous chapter is maximising the number of players in an interactive multiplayer game. We identify four classes of approaches that could be used to achieve this goal: changes to the Internet architecture so that routing considers performance (Section 3.1), application-layer mechanisms (Section 3.2), improvements to the game network infrastructure (Section 3.3), and pragmatic improvements to Internet routing (Section 3.4).

3.1 Clean-slate Internet Architecture Approaches

This section discusses a subset of prior works that can provide latency-aware routing through radical redesigns of the Internet's control plane or new Internet architectures.

Scalability, Control, and Isolation On Next-generation Networks (SCION) [12] is a clean-slate design of inter-domain routing that provides visibility of all feasible paths between two networks on the Internet to the hosts in networks. It provides a control plane protocol that discovers and distributes Autonomous System (AS)-level path segments that contain topology information about the links and interfaces connecting consecutive ASes within a path. These path segments are distributed to end hosts that can use them to construct

a forwarding path to specific destinations via the available paths. The control plane protocol provides a mechanism that allows the forwarding and performance information to be embedded in packet headers by the end hosts. This mechanism allows packets to be forwarded arbitrarily through a selection of path segments based on performance information, such as the latency on a path segment embedded in the packets.

Although SCION can provide latency-aware routing, it is impractical for the problem we intend to solve in this work for two reasons. First, the SCION control plane is incompatible with existing inter-domain infrastructure and cannot operate side-by-side. Furthermore, changes will be required to the network stack of end hosts in a SCION internet making large-scale adoption more improbable. Secondly, in client-server multiplayer games, the server must store and process significant path information for many clients. There is no evidence that the distribution of path segments would scale for a large network and not place a high storage burden on the hosts.

Pathlet routing [64] is an inter-domain routing protocol that enables an AS to advertise a sequence of virtual nodes (pathlets) along which the network is willing to route traffic to its neighbours. End hosts in other ASes concatenate these pathlets to form an end-to-end route to a destination from the selection of available pathlets. A path vector algorithm is employed to distribute the pathlets between neighbouring ASes, and additional labels can be attached to different pathlets from the same network to indicate different Quality of Service (QoS) levels on each pathlet.

Pathlet routing provides visibility over available routes and allows flexible traffic steering via these routes. However, in the context of maximising the number of players that can connect to a multiplayer game on the Internet, it cannot be adopted because the routing protocol it uses is incompatible with current inter-domain control plane.

New Internet Routing Architecture (NIRA) [158] is an inter-domain routing control plane implemented to allow users in stub networks to select

the sequence of transit providers their packets must traverse on the way to their destination. It provides visibility over available routes to stub networks and allows the end hosts in the network to choose the preferred egress routes for traffic out of their network. However, it proposes a network protocol incompatible with the Border Gateway Protocol (BGP).

Feedback Based Routing [163] is another inter-domain routing control system that operates similarly to Pathlet routing described earlier. It also provides the same capabilities as Pathlet routing.

Another Multipath Interdomain Routing (AMIR) [114] shares principles similar to NIRA. It provides multiple paths to stub networks and a mechanism for the networks to flexibly select their path to a destination.

There are also Software-Defined Networking (SDN) proposals of clean-slate interdomain routing solutions to provide Performance-Aware Routing (PAR) that the current inter-domain routing system lacks. Examples of such solutions are Software-Defined Inter-Domain Routing (SDI) [150], Multi-dimension Link Vector (MLV) [30] and Route Chaining System (RCS) [151].

This section examined clean-slate proposals and solutions that can provide latency-aware routing for multiplayer games. As discussed, most of the solutions provide latency-aware routing via different approaches. SCION, for example, enables packet forwarding through arbitrary paths using path information embedded in packet headers. The ability to embed path information in packet headers could make flexible forwarding of packets through arbitrary paths easy. However, the need for compatibility with the existing inter-domain routing control plane means these solutions cannot be practically adopted to provide latency-aware routing for multiplayer games.

3.2 Application-layer Techniques

Game developers employ various latency compensation techniques *at the application-level* to lessen the effects of latency. These techniques, running on the game

client or server, alleviate latency by adjusting player input, game actions, game state, and rendered output. According to [95], latency compensation techniques can be organised into four main classes: feedback, prediction, time manipulation, and world adjustment. Feedback techniques provide audible or visual information to players based on latency without altering the game state. Prediction techniques are often applied on the client side, enabling the client to estimate the game state without having the authoritative game state from the server. Time manipulation techniques involve altering the game world's virtual time to compute the game state or resolve player actions. World adjustment techniques modify the game state to decrease difficulty as latency increases.

Numerous multiplayer games use one or more of these techniques to counteract the impact of latency. For instance, RIOT Games has implemented a combination of feedback, prediction, and time manipulation techniques to minimise latency's effect in Valorant [39]. Similarly, in [86], a feedback latency compensation approach that visually masks latency, enhancing the perception of responsiveness, was employed. These techniques help mitigate and conceal the effects of latency on players to prevent them from disconnecting from the server and are complementary to the scheme proposed in this thesis.

3.3 Improvements to Game Network Infrastructure

Some online game companies have strived to improve gaming experience by enhancing the game network infrastructure as much as possible. For example, RIOT DIRECT [101] is a private Wide-Area Network (WAN) implemented by RIOT Games – a large game provider, to ingress/egress game traffic near their clients. RIOT games achieve this by peering with eyeball networks where the clients reside at multiple Points of Presence (PoPs) spanning different locations. Their peering routers at these PoPs are connected to the datacentre regions where their game servers are with the private WAN. This implementation

enables RIOT Games to keep game traffic in their network for as long as possible and minimise the transit time of their game traffic on the public Internet. In cases where they are peering directly with a client's network, the game traffic does not transit through the public Internet at all. This enables them to provide feasible latency for their multiplayer game application traffic.

RIOT DIRECT addresses the same problem this work focuses on and is complementary to our proposed solution. However, two issues with the RIOT DIRECT implementation limit its effectiveness. First, RIOT DIRECT can only benefit clients in regions and countries where they can establish peering at nearby PoPs with eyeball networks in those regions. Otherwise, game traffic between clients and their servers will be transmitted over the public Internet. Second, when RIOT DIRECT establishes multiple peering with a client's network at different PoPs, each providing different latencies to the client, it may be unable to steer traffic towards the PoP with the lowest latency.

Subspace [139] is an Internet Service Provider (ISP) that offers a WAN optimised for low latency to game providers. The WAN aims to achieve a similar objective as RIOT DIRECT: to minimise the transit time of game traffic on the Public Internet. Commercial Gamers Private Network (GPN) solutions such as WTFast [152] and ExitLag [44] provide similar services to game players instead of game providers. These implementations operate on a similar principle to RIOT DIRECT and have the same limitations.

3.4 Pragmatic Improvements to Internet Routing

In Section 3.4.1, we discuss a set of PAR solutions that are based on BGP Egress Peer Engineering (EPE), similar in principle to our proposed solution. Section 3.4.2 assesses the suitability of these solutions to the problem stated.

3.4.1 Performance-Aware Routing via EPE

EPE [53] is a technique that enables the arbitrary selection of policy-compliant routes to a destination from various peers and the directed routing of specific traffic to that destination via a particular peer. Solutions based on EPE enable flexible performance-aware inter-domain routing independently of BGP and Interior Gateway Protocol (IGP) policies in the network. This section discusses some of the EPE solutions that provide PAR.

RouteScout [7] is a hybrid software-hardware egress engineering control system that enables PAR in stub ASes. It dynamically adapts the routing of outgoing traffic across multiple paths to a destination using P4-enabled hardware. RouteScout tackles the challenges of continuous evaluation and monitoring of performance across multiple paths to a given destination, the allocation of traffic onto those paths and the level of granularity at which traffic is distributed on those paths. It exploits the programmability of P4-enabled hardware to implement probabilistic data structures in the dataplane. These structures enable the collection and aggregation of delay and loss measurements for TCP-based traffic flows on a per-destination granularity. Using the measurements collected from the dataplane, the RouteScout controller utilises linear optimisation heuristics to compute a new forwarding state that achieves traffic splitting across the paths as required. The RouteScout system is deployed at the edge of the AS and transparently splits traffic over policy-compliant BGP paths at the network's edge. However, RouteScout requires P4 programmable switches to be deployed in the network. Additionally, its data structures for collecting and aggregation delay and loss measurements are unsuitable for connection-less services, such as multiplayer online games, which typically use UDP.

Espresso [160] is an Internet peering edge routing architecture designed to provide PAR for a large content provider network. Google designed it to manage link utilisation and dynamically reroute traffic away from congested links to minimise loss. Using a specialised network fabric, Espresso enables

the servers responding to user requests to steer their responses to users via an alternate path without violating BGP policies. This system enables Google to shift and split traffic between multiple peers, edges, and links without triggering the BGP path selection mechanism.

Edge Fabric [123] is also an Internet peering edge routing architecture designed to provide PAR for a large content provider network. Facebook designed it to manage link utilisation and dynamically reroute traffic away from congested links. Unlike Espresso, Edge Fabric utilises existing vendor software and hardware to achieve capacity-aware routing while minimising the need for custom hardware and network redesign. Specifically, Edge Fabric relies on software and implementations such as BGP monitoring protocol (BMP) [125], Simple Network Management Protocol (SNMP) [50], and Internet Protocol Flow Information Export (IPFIX) [5] that are stable and standardised to provide inputs to the system. Edge Fabric steers traffic through a peering edge router to a specific destination by sending a BGP update with a high local preference attribute to the peering edge router. A centralised EPE controller with access to the routing tables of all peering edge routers sends the update. The Edge Fabric solution adopts a pragmatic approach to providing performance-aware inter-domain routing by utilising existing vendor software and hardware. This approach prioritises the ability to redirect traffic for a prefix between peering edge routers over the fine-grained routing control offered by solutions like Espresso and RouteScout, which require specialised hardware and network fabric.

COFFEE (Content Oriented Flexible Framework with Egress Engineering) [84] is another solution that employs EPE to provide PAR for content provider networks. It utilises passive measurements to calculate QoS scores for traffic flows across the available paths and steer them to a peer based on their scores. The COFFEE implementation enables content operators to proactively improve user Quality of Experience (QoE) for content being served from their network by guiding flows along the most suitable QoS

optimal path. This is accomplished by aggregating routes from all BGP border routers and providing them to the content servers. The content servers then use passive measurements to estimate the QoS of flows via these paths. COFFEE provides a mechanism that allows the end hosts, in this case, the content server, to choose the path for its traffic flow independently in response to changing conditions rather than relying on the network to make that determination. Essentially, COFFEE enables the end hosts to make routing decisions that align with the QoS requirements of their traffic flows. However, COFFEE's approach of rerouting flows due to latency spikes on a path has limitations in handling excessive latency resulting from routing changes and instability.

3.4.2 How Suitable are Existing PAR Approaches?

To assess the suitability of the existing PAR solutions to the stated problem, we ask three different questions:

1. **Is there visibility of available routes?** Multiple routes that can be used without compromising routing correctness are required to provide latency-aware routing for multiplayer games. These routes may not always be visible due to BGP's path-hiding behaviour. Existing solutions implemented mechanisms to collect and aggregate available routes to a destination to address this.

2. **Is continuous performance assessment possible?** The performance of the identified feasible routes and the traffic carried on those routes must be continuously monitored to determine the best route. In the multiplayer game context, feasible latency on paths between the server and clients is the relevant performance metric. Existing PAR solutions provided continuous monitoring for performance metrics specific to their context, i.e. congestion and loss.

3. **Is flexible traffic steering possible?** Traffic should be steered based on performance metrics collected about these feasible routes in a manner

that does not compromise routing correctness and stability. Some existing PAR solutions considered in this chapter required specialised network equipment and fabric to achieve this flexible traffic steering. In cases where specialised network equipment and fabric were not required, they provided flexible steering by changing the BGP path selection process, which would affect whole prefixes instead of a single client.

The BGP EPE solutions discussed in Section 3.4.1 provide PAR by steering traffic flows to specific egress peers without violating BGP protocol specifications or policies. However, these solutions have two limitations that make them impractical for the problem we intend to solve.

First, they require the use of specialised network equipment and fabric. RouteScout, for instance, requires P4-enabled hardware to provide PAR, and the probabilistic data structures used for collecting delay and loss measurements were implemented for TCP flows. Espresso, on the other hand, requires a specialised network fabric.

Second, they may affect routing correctness and stability if they provide latency-aware routing for multiplayer game applications. Edge Fabric overrides default BGP routing decisions to optimise link utilisation and avoid congestion by generating BGP updates with high local preference attributes for the preferred routes. Also, Espresso and Edge Fabric were both designed to avoid congestion and optimise link utilisation; as such, they are more suitable for improving routing decisions for TCP-based applications, which are typically sensitive to congestion. Furthermore, playing with the BGP path selection process to shift the traffic of whole prefixes to another path is too coarse for the specific requirements of multiplayer online games. Considering the need for faster and more specific changes required for multiplayer online games, this approach would likely cause route oscillations.

3.5 Conclusion

This chapter identified and discussed four types of prior works and solutions for minimising player disconnections due to latency in multiplayer online games. The first class includes clean-slate and novel Internet architecture solutions primarily conceived to address the absence of multipath and PAR on the Internet. However, these solutions are incompatible with existing inter-domain routing protocols and technologies, and their deployment cannot be implemented incrementally. The second class was application-level techniques that complement our proposed latency-aware routing approach for multiplayer games. In the third class, we discussed improvements to game network infrastructure that use private WANs to minimise the transit time of gaming traffic on the public Internet. Finally, we discussed PAR solutions based on EPE that are similar to ours.

In discussing similar PAR solutions, we asked three questions to assess the suitability of these solutions to the problem stated and identified two limitations that make these solutions unsuitable for our work. Across the various types discussed in this section, three key limitations make them impractical for our purposes. The first limitation is the incompatibility of clean-slate solutions with existing inter-domain routing control plane. The second limitation is the requirement for specialised hardware and specialised network fabric. The third limitation is the inability of these solutions to guarantee routing stability and correctness if applied to provide latency-aware routing for multiplayer games.

Chapter 4

Overwatch Design and Architecture

This chapter presents the design and architecture of Overwatch, a pragmatic latency-aware routing control system proposed to confirm our hypothesis. It is a Border Gateway Protocol (BGP) Egress Peer Engineering (EPE) solution that provides latency-aware routing for interactive multiplayer online games.

The structure of this chapter is as follows: Section 4.1 discusses the principles that guided our design of the system. Section 4.2 explains the choices made to enable the practical implementation of latency-aware routing based on those principles. Section 4.3 provides an overview of the solution's architecture, aligning it with the previously discussed design choices and principles. The chapter wraps up with a concluding discussion.

4.1 Design Principles

Overwatch is a pragmatic Performance-Aware Routing (PAR) control system designed to provide more efficient routing options for latency-restricted inter-domain traffic in an Autonomous System (AS). The design of Overwatch prioritises practicality, routing stability and minimal changes to provide modest gains. To achieve this, Overwatch adheres to six design principles discussed below.

Guarantee/respect path legitimacy. Overwatch aims to steer traffic to a destination arbitrarily via any egress peer. However, steering traffic via peers or upstreams that have not announced routes to a destination can lead to unintended or abusive traffic *detouring*[1], as discussed in [105]. This *detouring* behaviour may be counterproductive to the goal of optimising inter-domain routing for traffic because ASes receiving the detoured traffic may apply packet filtering policies, in addition to AS path and prefix-based filtering in BGP, to prevent this from happening. Therefore, Overwatch must ensure that it exclusively utilises policy-complaint routes that have been appropriately announced to the network for steering traffic to a destination.

Minimal change/Incremental deployment. Other solutions may require specialised hardware, software, and network fabric. However, Overwatch should use existing hardware, software, and standard Internet practices. This approach eliminates the need for custom hardware or protocols. By integrating with the current software and routing equipment, Overwatch ensures that minimal logical and physical modifications to the network are required. This results in less effort to introduce the solution. Furthermore, Overwatch operates within a single AS and should not require cooperation from other ASes to implement its routing decisions.

Reactiveness. Internet routing information changes on a daily basis. Also, network quality can deteriorate, and traffic conditions can fluctuate. When this performance variability is detected, Overwatch should be able to dynamically respond to these changes and reroute flows to a better path.

Compatibility with BGP. A network running Overwatch needs to interoperate with the rest of the Internet and neighbour ASes to exchange routing information and maintain connectivity with the rest of the Internet. Therefore, Overwatch should maintain compatibility with BGP and refrain from disrupting established inter-domain routing practices that could lead to routing instability.

[1] detouring is the sending of packets to any egress peer regardless of received prefixes.

Limited per-flow flexible routing. Destination-based forwarding and longest prefix matching on the Internet lack flexibility in mapping packets across multiple paths and can conflict with other Traffic Engineering (TE) objectives [74]. The conflict of this forwarding approach with TE objectives can jeopardise routing correctness and stability if not properly managed. Some existing solutions optimise inter-domain routing by announcing more specific prefixes or overriding BGP's decision to steer traffic via a preferred path. However, this practice contributes to the growing size of the Internet routing table, posing a potential threat to long-term stability. As such, Overwatch should provide limited per-flow traffic steering to game clients without using BGP to alter the forwarding path for the game traffic flow.

Minimise impact of BGP routing policy changes. BGP routing changes can have a significant impact on network performance. Sometimes, it can result in up to 30% packet loss, which can persist for up to two minutes because of a single BGP routing change [113, 148, 87]. However, some prior works induce BGP routing changes by modifying BGP path attributes to implement new policies for steering traffic optimally. Nonetheless, due to BGP's inherently slow and chatty convergence process, this approach is ill-suited for interactive real-time applications. Overwatch should aim to mitigate the adverse effects of BGP convergence on the traffic being steered by isolating the dataplane and updating the dataplane only when necessary.

4.2 Design Choices

This section discusses three design choices that adhere to the principles outlined earlier.

1. **Use Policy-compliant Routes**: Overwatch strictly uses policy-compliant routes for steering traffic to a destination. Networks may desire distinct traffic from different neighbouring networks. As a result, they may apply inter-domain routing policies that could conflict with their neighbours

and compromise routing correctness and stability. This is because BGP enables networks to define and implement these policies independently. Considering this, using paths that contravene other networks' policies may result in *detouring*, which will be counterproductive.

Within the Overwatch architecture, policy-compliant routes are obtained by aggregating and centralising the Routing Information Base (RIB) from all external BGP (eBGP) devices in the network. The routing extraction information subsystem discussed later enables Overwatch to receive route announcements from BGP devices in the network after each device has processed the external routes it has received. By adopting this approach, inter-domain routing policies can continue to be applied independently to each eBGP device in the network, consistent with BGP protocol specification. At the same time, Overwatch steers traffic via these routes. While this approach may reduce the number of possible routes that can be used, it guarantees that the routes used are policy-compliant and maintain consistency at the inter-AS and intra-AS levels. This, in turn, prevents undesired behaviours, such as detouring or traffic blackholing.

Secondly, Overwatch ensures that its inter-domain route optimisation decisions are transparent to external networks and BGP. Overwatch achieves this by incorporating a controller subsystem discussed later that furnishes precise forwarding instructions to a specific node to guide traffic via the optimal path. This flow steering is accomplished through source routing – which enables a node in the network to specify the intermediate points that traffic must traverse to reach its destination. Crucially, implementing flow steering functionality is only necessary on the node receiving instructions from the controller. Given that Overwatch uses policy-compliant routes, this mechanism enables it to specify the path to reach any egress peers announcing the policy-compliant routes. This happens without overriding BGP's best path decision or fragmenting

prefixes across multiple paths.

2. **Use Commodity Hardware and Standardised Software**: Overwatch takes advantage of existing and standardised software implementations, hardware, and protocols to monitor the performance variations of paths and respond dynamically to these changes in conditions using limited, flexible per-flow steering capabilities in the dataplane of nodes. In contrast, some solutions discussed in the previous chapter require specialised hardware and network fabric. The per-flow steering functionality can be implemented on edge network devices, intermediate devices, or end hosts using existing hardware and software implementations within the Overwatch architecture. The network fabric does not have to be altered to provide this per-flow steering functionality. Additionally, the latency measurement subsystem in the architecture can use existing network monitoring tools and management to monitor the paths and collect performance information.

3. **Ensure Compliance with BGP Specification**: Collecting and exchanging routing information with BGP is vital in optimising inter-domain routing. Solutions requiring changes to the protocol specification are unlikely to gain widespread adoption. This is because, while the BGP protocol specification has seen revisions and updates, ensuring backward compatibility remains paramount. All BGP appliances must adhere to the same or nearly the same version of the protocol for proper operation. In the Overwatch architecture, the routing information extraction module offers an interface that allows the system to exchange routing information with BGP devices in a protocol-compatible manner.

4.3 Architecture

Overwatch is a routing control system whose design and implementation use EPE alongside existing IP protocols to provide more efficient routing options for latency-restricted inter-domain traffic. The system employs source routing

to steer flows from an ingress node to the optimal egress node, ensuring that specific traffic is routed through the best egress path for the destination. This occurs independently of underlying BGP and Interior Gateway Protocol (IGP) routing policies. The Overwatch architecture nominally consists of four main components. The *routing information extraction* subsystem maintains a topology database for intra-domain routing information and serves as a BGP speaker for exchanging inter-domain routing information with eBGP devices. The *measurement* subsystem continuously assesses the performance of the paths to a destination, using active or passive measurement approaches as required. The *controller* subsystem executes the algorithm responsible for computing the routing decisions and forwarding instructions. These instructions are used to configure the dataplane of the ingress node, enabling the steering of traffic along the path that provides feasible latency for the flows. An illustration of the architecture in a game provider network is shown in Figure 4.1. Both routing and controller subsystems operate on commodity hardware, allowing for scalability in the network as necessary. Finally, the *flow steering* subsystem, a source routing dataplane, can be implemented on commodity hardware. Each subsystem is described below.

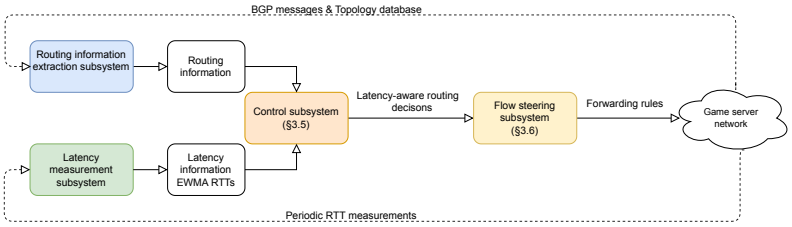


Figure 4.1: Overwatch routing control system.

4.3.1 Routing Information Extraction

The routing information extraction subsystem serves two purposes: (i) enables the centralised view and aggregation of the routing table of eBGP devices in the network, explicitly providing this information to the controller subsystem;

(ii) use topology data from IGPs to aid the routing decisions for optimised inter-domain routing.

To accomplish these objectives, the routing information extraction subsystem maintains a topology database for intra-domain routing information. It also leverages BGP messages for exchanging inter-domain routing information with eBGP devices across the network. The ability to exchange BGP messages with the eBGP devices in the network enables Overwatch to receive, reflect, and manipulate inter-domain routes. This is achieved without modifying external peers, protocol extensions, or extensive reconfigurations. Furthermore, using BGP messages supports an incremental and minimalist deployment approach. In this approach, only a subset of the eBGP devices in the network must be configured to exchange routing information with Overwatch to provide efficient routing options.

Consolidating inter-domain routing data. Fundamentally, the crucial role of the routing information extraction subsystem is collecting inter-domain routing information, enabling Overwatch to make efficient and informed routing decisions. This can be achieved via different means. Three broad approaches are discussed, and the choice used in the architecture is highlighted.

In the first approach, all eBGP devices are modified to provide an interface to collect and send all routing information in their Adj-RIB-Out table to Overwatch. However, in this approach, an additional control plane communication interface with the eBGP devices is still necessary to manipulate the inter-domain routes received from these devices. An API, such as the one specified by λBGP [73], could be employed, but this would entail customisations and alteration to the eBGP devices in the network. Additionally, it would necessitate increased control plane signalling with the eBGP routers, thereby introducing scalability concerns.

The second approach involves taking a dump of the RIB of eBGP devices using the BGP Monitoring Protocol (BMP) [125]. While this method provides a comprehensive view of all available routes per device, it does not explicitly

indicate the best path chosen by each device. Consequently, Overwatch would have to emulate the BGP best path selection algorithm on the RIB of each device, which can be complex due to the numerous tunable parameters that can be configured on devices to alter the path selection process. This emulation process would introduce the potential for divergent decisions between Overwatch and the eBGP devices, potentially causing routing instability.

The third approach is based on establishing peering sessions with eBGP devices to collect routing updates. This has the benefit of not requiring customisations to the eBGP devices in the network. On the other hand, it offers a more limited view of the RIB if the BGP *ADD-PATH* [146] – a capability that allows the advertisement of multiple paths is not used, as only the best paths are exchanged through BGP update messages. However, this approach guarantees that Overwatch only considers paths actively present in the forwarding table rather than relying solely on paths in the RIB. This minimises the risk of using non-policy-compliant routes. The routing information extraction subsystem adopts this approach. The implementation is further discussed in the next chapter.

4.3.2 Measurement

The goal of the measurement subsystem is to continuously assess the performance of the paths to a specific destination. Typically, accurately measuring the performance of arbitrary paths on the Internet can be quite challenging due to the need for path control and vantage points. The performance of the paths to a destination can be estimated via passive or active means. Overwatch has control over traffic routing within its network. It can use this control to direct probing traffic along specific paths or steer some traffic onto alternative paths to access performance.

Frequent measurements are necessary for an accurate assessment of the performance of paths. However, active measurements take time and may be prone to estimation errors. Additionally, they typically rely on sampling

rather than continuous monitoring, which can be costly. Therefore, if active measurements are employed, the frequency of probes can be adjusted according to the scenario considered.

The latency measurement subsystem is implemented to (i) demonstrate that these measurements can be achieved using simple and readily available tools, and (ii) showcase that the measurement samples can be adapted and adjusted according to the specific scenarios considered. The implementation is discussed in the next chapter.

4.3.3 Controller Subsystem

The Overwatch controller subsystem synthesises inputs from other components in the Overwatch Architecture to make efficient routing decisions. It operates the algorithm used to make these routing decisions and generates forwarding instructions based on those decisions for the dataplane to steer traffic accordingly. Specifically, the controller subsystem receives routing information from the routing information extraction subsystem and network performance metrics from the measurement system. The algorithm executed by the controller can be easily customised and configured as required to achieve specific PAR objectives, provided that the correct inputs are available. For instance, the algorithm can be provided to reroute particular traffic flows to a set of destinations to paths with lower latencies or to circumvent specific paths or upstreams. With the inputs provided to the algorithm, the controller computes a routing decision for these flows and generates forwarding instructions to configure the dataplane, ensuring the desired routing objectives are met for the traffic flow.

In addition to optimising routing for specific traffic flows based on performance objectives, the controller subsystem takes advantage of the expanded Random Access Memory (RAM) capacity that is available on commodity hardware. This allows it to define and store an arbitrary number of routing tables and filters to enable flexible selection of inter-domain routes. Each routing table can hold multiple routes from different eBGP peers to a destination. Furthermore,

each table can be configured with an explicit list of peers allowed to import or export routes from that table. This provides fine-grained control over the routes that can be advertised to each peer. Achieving this with existing internal BGP (iBGP) schemes, such as route reflectors, confederation, or full meshes is not trivial.

For instance, if there are three distinct paths learnt from three different eBGP peers in the network, the controller can be configured to ensure that different peers receive a customised announcement of a subset of the available routes. Depending on the iBGP schemes employed in the network, achieving this level of customisation is either impossible or would introduce additional management complexity and scalability concerns. In addition to the multiple routing tables, the controller subsystem also provides fine-grained access control lists. These can be used to determine which route is exported or imported into the tables or announced to peers based on any of the route attributes. The implementation of the controller and algorithm is described in the next chapter.

4.3.4 Flow Steering Subsystem

The flow steering subsystem enables specific traffic flows to be egressed through more favourable paths to achieve the required performance objective without routing inconsistencies. This dynamic and fine-grained flow steering cannot be achieved with BGP and conventional intra-domain routing protocols, as they generally rely on destination-based forwarding. Although relatively simple, destination-based forwarding is limited and does not offer a flexible method for mapping packets over multiple paths [74]. Therefore, an alternate approach is required to "customise" how specific packets are steered and forwarded from one point to the other.

In the Overwatch architecture, source routing is the alternative approach used to achieve this flow steering capability. It allows a "source" node to specify, either partially or completely, the desired route that a packet must traverse through the network. Source routing can be accomplished through

various mechanisms. Dataplane implementations, such as Multiprotocol Label Switching (MPLS) [145] and OpenFlow [102], can be used to direct flows at the dataplane level. However, it is important to consider the trade-offs associated with these approaches. While MPLS is the most commonly deployed of all the options, MPLS-based approaches have a high overhead because they need to maintain an explicit state at all hops along the MPLS path, which may limit control and dataplane scalability [45]. On the other hand, OpenFlow may necessitate modifications and upgrades to underlying equipment, which can impose additional costs and operational challenges.

Segment Routing (SR) [54] implements source routing in the Overwatch architecture. It is a source routing approach that enables networks to guide packets along any desired path by embedding an ordered list of instructions, known as segments, within packet headers. These segments specify the list of waypoints and intermediate hops the packet must traverse to reach its destination. In contrast to previously discussed mechanisms, SR offers the ability to steer traffic flows along traffic-engineered paths without requiring per-flow state maintenance at each node.

SR can be realised through two dataplane implementations: Segment routing over MPLS(SR-MPLS) [13] or Segment Routing over IPv6 (SRv6) [52]. SR-MPLS can be implemented using an MPLS forwarding plane with no change in behaviour [13]. In contrast, explicit routing in SRv6 is achieved using an IPv6 dataplane. The *headend*[2] appends the segments to the packet headers in both implementations to achieve explicit routing. A segment is defined by a Segment Identifier (SID); in SR-MPLS, it is an MPLS label, while for SRv6, it is an IPv6 address. SR stands out as a dataplane choice due to its compatibility with widely-used intra-domain protocols, such as IS-IS, which allows SIDs to be distributed and processed on well-matured commercial routers equipped with protocol extensions [112, 143]. Overwatch was based on SRv6 pragmatically due to the availability of open-source software implementation, which was used

[2]The node where segments are written into the packets. In this approach, it could be the game server or router between the server and the edges of the network.

in the prototype. The implementation of the flow steering will be discussed in the next chapter.

4.4 Summary and Discussion

By leveraging the high storage capacity RAM available on commodity hardware and capitalising on the flexible flow steering capabilities provided by SR, Overwatch aims to deliver pragmatic inter-domain latency-aware routing for multiplayer online games.

The architecture of Overwatch is designed to ensure routing correctness and stability by enforcing the use of policy-compliant routes for efficient routing. At the same time, it aims to maintain compatibility with BGP. Overwatch accomplishes this by transparently separating the forwarding policy for specific traffic flows from BGP and IGPs's best path forwarding decisions. It does so by employing an SRv6 dataplane that requires minimal changes to existing infrastructure and allows for incremental deployment. The flexibility offered by the SR dataplane in Overwatch provides the following benefits:

1. It enables the system to dynamically respond to excessive latency on a path and swiftly redirect traffic from the game server to clients via a path with feasible latency.

2. It allows for the incorporation of additional policies and network intents to provide more efficient routing options for inter-domain traffic.

3. It mitigates the impact of BGP policy changes on multiplayer game traffic.

Chapter 5

Overwatch Implementation

This chapter discusses the implementation of the Overwatch architecture, designed to enable latency-aware routing for multiplayer games. Section 5.1 explains how the routing information extracted subsystem is implemented to collect and process routing data from Border Gateway Protocol (BGP) devices. Section 5.2 discusses the implementation of the latency measurement subsystem. Section 5.3 presents the implementation of the controller subsystem and algorithm that makes the latency-aware routing decisions. Also, the section explains the operation of the latency-aware routing algorithm. Section 5.4 describes the flow-steering implementation using a Linux-based Segment Routing over IPv6 (SRv6) node.

The four subsystems in Overwatch were developed using Python and various open-source applications and tools. They were designed to be configurable and deployable on commodity hardware. The source code for the prototype is accessible at [47]. In the prototype implementation, the subsystems can operate on separate hardware or be consolidated on a single hardware. This demonstrates its scalability to accommodate growing network needs and support for incremental deployment.

5.1 Routing Information Extraction Implementation

The primary objective of the routing information extraction subsystem is to aggregate and consolidate BGP updates. This aggregation of routing information enables the controller to make informed latency-aware routing decisions for flows. ExaBGP [97] is used to aggregate inter-domain routing information in Overwatch. It is a tool that allows applications and services agnostic to BGP to exchange BGP messages programmatically with eBGP devices in a network. ExaBGP functions as a BGP speaker that interfaces between the rest of the Overwatch subsystem and eBGP devices in the network. To facilitate the exchange of BGP messages, gRPC [67], an open-source Remote Procedure Call (RPC) framework, is used to build a client-server API that allows BGP messages to be delivered to the controller subsystem process and vice versa. For efficient serialisation of BGP messages from ExaBGP and route updates from the system to eBGP peers, Protocol Buffers (protobuf) [68], a language-neutral and cross-platform data format, is relied upon.

5.1.1 ExaBGP Modules

ExaBGP and the controller subsystem's main process interact through a gRPC channel session established over TCP. Two Python application modules implemented for ExaBGP are responsible for processing, sending, and receiving messages via the gRPC channel session. The first Python module, which is the receiver module, collects and transforms the raw BGP messages from ExaBGP into protobuf data structures that can be easily extracted and parsed by the controller. This module is triggered whenever ExaBGP receives a BGP message from any eBGP peer. It serialises the BGP message with the protobuf formats and transmits the message via the gRPC channel session to the main process of the controller subsystem. Each BGP message type received from the eBGP peers is processed differently by the controller subsystem and is

used to update the controller's network view. The actions and state changes associated with the four BGP message types are discussed below.

1. **Open Message:** Overwatch uses the BGP open messages received to discern the capabilities and features supported by the eBGP router. For instance, based on the capabilities and features supported by the eBGP peer, the controller can determine if the peer supports different address family types. This will determine whether the controller can announce or receive different address families from the peer. If the route refresh capability is enabled, the controller can request that an eBGP peer resend all its routes following a reconfiguration. The hold timer value in the message determines how long the controller should wait for messages from a peer before it assumes that the peer is dead and begins marking routes from the peer as unusable and invalid. Finally, the Autonomous System (AS) number and BGP identifier are compared against the configuration intent specified in a configuration file for Overwatch to determine whether the controller should accept or share routes with an eBGP peer.

2. **Notification Message:** Overwatch interprets the notification messages received from eBGP peers to assess the state of the BGP session between the peers and the ExaBGP instance. Like the hold timer value in the Open message described above, Overwatch monitors these messages to prevent routing inconsistencies. It ensures that routes are exchanged only with peers in the established BGP state. It also marks routes from peers not in an established BGP state as invalid, so they are not used for flexible routing.

3. **Keepalive Message:** Overwatch parses and monitors the keepalive messages from the eBGP peers to ascertain their continued presence and connectivity. The controller subsystem maintains a timer that resets each time it receives a keepalive message from a peer. It marks a peer

inactive when it has not received a keepalive message within a duration exceeding the hold timer value. Marking a peer as inactive prompts the controller subsystem to label the routes from that peer as invalid and issue withdrawal messages to other peers for those routes.

4. **Update Message:** This is the most significant BGP Message processed by Overwatch. It contains the routing information essential for making latency-aware routing decisions. When Overwatch receives an update message from a peer via ExaBGP, it parses the message to determine whether Network Layer Reachability Information (NLRI)[1] are being announced, withdrawn, or a combination of both for some NLRI. If the update message solely announces some NLRI, the path attributes from the message are collected, along with the NLRI, for further processing by the controller. This involves adding it to the controller's routing table(s) and reflecting it to other peers per the configuration. In cases where the update message withdraws specific routes, the NLRI is extracted from the message and relayed to the controller subsystem to remove the routes from the routing tables. This action also triggers a withdrawal update message to other peers regarding the route.

Second ExaBGP Module. The second Python module is the sender module, and it acts as a translator of sorts for Overwatch. This module enables the system to send BGP messages to the eBGP peers. When Overwatch needs to announce or withdraw a route from one or more peers, it utilises the protobuf data format to serialise the data required by ExaBGP to encapsulate a BGP update message. For route announcements, the data includes information such as the NLRI, the address of the receiving peer, and route attributes such as Origin, next-hop, local preference value, and AS path. Data for withdrawals will include details such as the NLRI, next-hop, and address of the receiving peer. The sender module deserialises the protobuf message received from the controller. It then extracts the data required for ExaBGP to generate and send

[1]Network destinations such as IP address ranges and prefixes.

a BGP update message to withdraw or announce the NLRI to the receiving peer(s).

The implementation of the routing information extraction module offers flexibility by allowing ExaBGP to operate on separate hardware, thus supporting incremental deployment. A single ExaBGP application can establish peering connections with one or more eBGP devices as needed. This approach aids scalability as additional ExaBGP applications can be introduced over time to meet the expanding demands of the network. The client-server API-based routing information extraction enables Overwatch to maintain a *centralised view* of inter-domain routing information across the network. This, in turn, enables Overwatch to influence inter-domain routing decisions without requiring modifications or extensions to the BGP protocol.

5.2 Latency Measurement Subsystem Implementation

The latency measurement subsystem in the Overwatch prototype is implemented to estimate the latency between the application server (in this case, the game server) and a set of clients using paths via different transit providers and peers. Accurately measuring the latency of arbitrary paths on the Internet is very challenging. This is due to the need for path control and the absence of vantage points. This implementation demonstrates how the measurement of a limited set of clients remote to a network can be done by leveraging path control from inside the network.

Measurement orchestration. The measurement subsystem consists of a measurement orchestrator module responsible for launching latency measurement processes for each path. The Round-Trip Time (RTT) measurement via all paths to a client from the server is initiated when the client establishes a connection with the server. Upon connecting to the server, a client sends the first packet to the server, indicating that the client is operational and

configured to receive both application and measurement traffic on specific ports. This communication may occur within the same server application or as a separate module. When the measurement orchestrator module receives the packet from the client, it extracts and records the client's IP address and UDP port address from the packet header.

Probing clients on paths. The latency measurement module uses the client address collected to generate and send probe packets to the client periodically via all "available" paths. The probe packets are UDP packets containing a sequence number, the timestamp of when they were sent, the client's address, and the address of the latency measurement process responsible for sending the probe packet. The client returns the packets to the address of the latency measurement process, which can be integrated into regular in-game traffic during routine game client updates. Upon receiving the reply from the client, the latency measurement process calculates the RTT for that path by getting the difference between the sent and received timestamp of the packet. Including the sequence number and timestamp in the probes assists the latency measurement process in accurately calculating the RTT for each packet and setting a timeout for waiting for the client's return packet.

Processing and delivering measurements. The latency measurement process computes an Exponentially Weighted Moving Average (EWMA) of the RTT and Deviation of RTT (devRTT). It periodically sends these values to the measurement orchestrator through a Unix socket. The measurement orchestrator, in turn, aggregates the estimated RTT for all connected clients across available paths and forwards this information to the controller subsystem via a Unix socket.

A message format was designed using protobuf to facilitate the exchange of measurement data between these subsystems. Details of the schema for the protobuf message exchanged between the subsystems can be found in Appendix A. It is worth noting that the choice of Unix socket is specific to the implementation due to the evaluation environment used. However, Internet

Protocol (IP) sockets could also be employed if the different subsystems are distributed across separate machines.

Implementation choice. The measurement subsystem was implemented in Python, and the advantage of utilising a socket and protobufs for exchanging the information is that it allows for the possibility of implementing the various components of the measurement subsystem in different programming languages, thus improving interoperability.

Path control for probe packets. Each latency measurement process for a path is assigned a unique IP address in this implementation of the latency measurement subsystem. Subsequently, the controller subsystem is configured to create and enforce policies that classify and steer measurement traffic from each latency measurement process via a designated path in the flow steering subsystem. The details of how this traffic classification and steering of traffic are accomplished will be elaborated upon in the subsequent section dedicated to the flow steering subsystem.

Measurement frequency and approach. The frequency at which the measurement subsystem sends probe packets can be adjusted according to specific scenarios. Higher-frequency transmission results in greater overall system accuracy and responsiveness (the effect of this interval adjustment is discussed in the evaluation chapter). The measurement performed by this implementation relies on sampling as opposed to continuous monitoring, which can be resource-intensive for the network.

Measurement inaccuracies. In practice, measurements take time, and a periodic sampling approach like the one implemented here can be susceptible to estimation errors. The time it takes for the actual measurement (i.e., RTT) between sending and receiving probe packets is typically small compared to the chosen sampling interval. However, the cumulative time taken for the measurement processes within each interval, plus the interval itself, can accumulate over time due to CPU interruptions in the operating system. This can lead to inaccuracies and time drift.

To mitigate this, the latency subsystem is implemented to ensure that a measurement sample is collected, processed, and sent within each interval rather than waiting till the end of an interval. Furthermore, instead of sending the actual RTT values measured by the latency measurement subsystem, this implementation utilises the EWMA and deviation of the RTT, which are transmitted to the controller. The evaluation chapter delves into the impact of various parameters used in estimating the RTT.

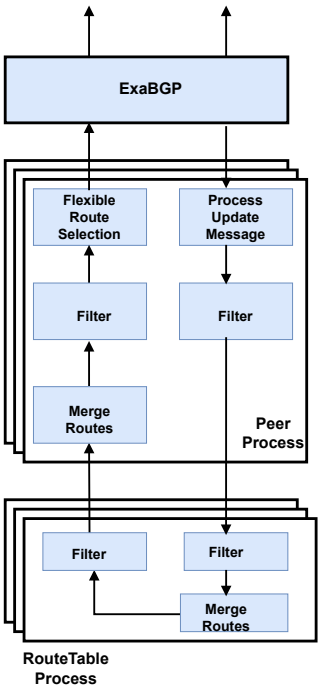


Figure 5.1: Overwatch's route processing pipeline: From BGP updates through flexible route selection to peer announcements.

5.3 Controller Subsystem Implementation

The controller application and latency-aware routing algorithm were implemented in Python, adopting the Actor model [4, 76] for concurrent computation in distributed systems. The Actor model offers a flexible approach where Actors are self-contained and concurrent entities. Each Actor has its own private state and message mailbox in this model, allowing asynchronous message exchange with other Actors. The state of an Actor can only be modified by processing

messages received from other Actors, and an Actor can transition the state or behaviour of another Actor by sending a message to that Actor. Messages sent to an Actor's mailbox are processed in a first-in, first-out order. Consequently, each Actor can carry out its tasks concurrently and independently without being blocked while waiting for a response from another Actor.

5.3.1 Controller Processes

In the controller application, multiple processes are created based on the configuration, taking into account the number of eBGP peers connected to Overwatch, the number of routing tables to be maintained by the controller, the traffic flows requiring latency-aware routing, and other system settings. Each process in the controller application operates as an Actor. The main controller process serves as the parent Actor, responsible for overseeing and initialising all other Actors according to the configuration.

ExaBGP process. For each ExaBGP instance, a BGP speaker process is launched. This BGP speaker process is responsible for receiving all BGP messages from the associated eBGP peers linked to the ExaBGP instance. It parses these messages as detailed in Section 5.1 and forwards them to the mailbox of the corresponding eBGP peer process (discussed shortly) for further handling.

The BGP speaker process also receives route update messages from the eBGP peer process. It serialises these messages according to the protobuf schema, indicating whether they are route announcements or withdrawals. These serialised messages are then sent to the sender module, as discussed in Section 5.1, to generate the BGP update messages intended for the corresponding eBGP peer devices via ExaBGP.

Peer processes. A dedicated peer process is established for each eBGP peer device connected to ExaBGP to provide routes to Overwatch. This peer process receives BGP updates forwarded to ExaBGP by the corresponding eBGP device. Upon receiving these updates, the peer process applies the

import and export filters specified in the configuration to the routes in the update message. Subsequently, it constructs a message containing the filtered routes and sends this to the designated routing table processes (described below), as defined in the configuration. As previously discussed in Section 4.3.3, the filters defined for the controller provide fine-grained access control lists, allowing the determination of which routes are accepted or rejected by a peer or routing table process.

Routing table processes. The system can have one or more routing tables, each implemented as a separate process. These routing table processes maintain routes received from various peers. They also receive messages from peer processes instructing them to add, remove, or update route entries. Consequently, a routing table can store multiple routes to a destination sourced from different eBGP devices.

Route import and export. The routing table can be configured to accept routes from specific peer processes based on the import filters in the configuration. Similarly, it can be configured to export routes to specific peer processes guided by export filters. When a routing table process receives a message from a peer process, it applies the import and export filters before relaying the message to the peer processes authorised to export routes, as well as to the latency-aware routing process.

Import and export filters. It is important to note that the routes in the messages sent by the routing table process to the peer processes may undergo further filtering at the peer processes themselves. A peer process applies additional filters upon receiving a message from a routing table process containing routes. It then selects a route to a destination based on the configuration's criteria.

Route exchange with eBGP peers. The selected routes are serialised into BGP messages and dispatched to the ExaBGP instance. The ExaBGP instance, in turn, generates the corresponding BGP route announcements or withdrawals for the respective eBGP peer devices. This capability enables

Overwatch to make informed decisions and influence the selection and announcement of routes to eBGP peers from other eBGP devices in the network. Figure 5.1 illustrates how Overwatch receives and sends routes to the eBGP peers via the ExaBGP instance.

Processing and retrieval of route information. To enable the efficient storage and processing of the routes received by the different processes in Overwatch, a C extension was implemented to provide optimised data structures for the routes and prefixes received by Overwatch from eBGP peers and used by the processes. This C extension also includes methods for accessing and updating these data structures.

Each route and prefix entry is stored as a C struct object with the C extension. Each struct member represents the attributes and properties of the route or prefix. For example, when an eBGP peer announces a route for a prefix, it is converted into a route entry, which includes the announced prefix and the associated attributes. Similarly, the prefix itself is transformed into a prefix entry, which encompasses the address of the prefix, its prefix length, and an indicator for the address family. The C extension module enables Overwatch to take advantage of the substantial memory capacity available on commodity hardware, enabling the storage of large volumes of route entries while ensuring efficient retrieval when needed.

5.3.2 Latency-Aware Routing Algorithm

The latency-aware routing algorithm is implemented as a separate process within the Overwatch system. The latency-aware routing algorithm was implemented in this Overwatch prototype to allocate server-to-client traffic to a path via an egress point. In line with the performance requirements highlighted in Section 2.1, its primary objective is to "bring" as many clients as possible to the game server with feasible latencies.

Since Overwatch uses the Actor model for the controller, it allows running multiple instances of the latency-aware routing algorithm to handle different

applications and traffic flows. Consequently, the algorithm needs to be straightforward and adaptable.

Inputs to the algorithm. To accomplish its goal, the algorithm redirects egress traffic for clients experiencing latencies near or exceeding a predefined application threshold to paths with lower latencies whenever such paths are available. It takes the set of clients and their end-to-end latencies as input, determining which clients should be shifted and to which path. These inputs are collected from messages containing available routes to the clients received from the routing table process(es) and the route selected by a peer process for the client prefix. Furthermore, the algorithm receives topology information from a topology database maintained by the main controller process and measurements from the latency measurement subsystem.

Algorithm output. With these inputs, including measurements, topology and routing information, the latency-aware routing algorithm generates a segment list (discussed in the next section below) that is then applied to the dataplane of the headend node by the main controller process. This action directs traffic flows along different paths.

Additional algorithm use cases. The algorithm provided functions effectively for the gaming scenarios, as will be demonstrated in the evaluation. However, it can be enhanced or customised to suit specific use cases or applications. For instance, the control algorithm could use topology information to proactively shift players when latency begins to rise and better path(s) become available.

Algorithm parameters. The algorithm whose pseudocode is given in Algorithm 5.1 uses the following variables. T is the *routing period*, representing the interval at which the controller evaluates the latency on paths and makes a decision. (The algorithm does not explicitly refer to the measurement period, which is M, $M<T$.) r is a route to c, a client and denoted $c.r$. Variable $c.r.RTT$ holds the latest estimated latency between the server and client c via route r, while $c.r.devRTT$ has the corresponding deviation in measurements. The set $c.routes$ contains routes between the server and client c, whereas

c.curroute indicates the current route in this set (initially set to the path selected by BGP according to the routing information extraction subsystem when the controller first learns about this client). There are two constants: MAL is the maximum acceptable latency, which is application-defined, and K (≥ 1), a factor used with the RTT estimate deviation to provide a "safety margin". The use of K helps clients on routes with latencies marginally under MAL but which may intermittently exceed it. The latency measurement subsystem continuously shares with the controller values for $c.r.RTT$ and $c.r.devRTT$ for each r in $c.routes$, doing this at fixed intervals.

Algorithm 5.1: Controller algorithm for latency-restricted routing

```
Require:
 1: T : Routing period for evaluation
 2: MAL : Maximum acceptable latency threshold
 3: K : Safety margin factor (K ≥ 1)
 4: clients : Set of all clients
 5: For each client c:
 6:    c.routes : Set of available routes to client
 7:    c.curroute : Current route (initially BGP-selected path)
 8:    For each route r in c.routes:
 9:       c.r.RTT : Latest estimated latency
10:       c.r.devRTT : RTT measurement deviation
Ensure:
11: Updated segment list for game traffic flow redirection
12: Optimal route selection for clients exceeding MAL
13: // Initialise current routes to BGP-selected paths
14: for each client c do
15:    c.curroute ← BGP-selected path from routing information
16: end for
17: for each T do
18:    for each client c do
19:       if (c.curroute.RTT + K × c.curroute.devRTT ≥ MAL) or c.curroute.RTT = 0 then
20:          f = ∅
21:          for r in c.routes do
22:             if c.r.RTT < MAL then
23:                f = f ∪ r
24:             end if
25:          end for
26:          if f ≠ ∅ then
27:             c.curroute ← min(f, key = λ r: c.r.RTT)
28:          end if
29:       end if
30:    end for
31: end for
```

Algorithm operation. Algorithm 5.1 operates periodically according to the routing period, during which latency measurements are provided (asynchronously). In each cycle (L1-15), the algorithm checks if a client's (L2-14) latency is excessive (L3), also considering an error margin. If the algorithm detects excessive latency, it tries to remediate it by choosing an alternate route (L4-13). The algorithm determines which routes to this client are feasible and saves

the result in a set (L5-9). If one or more routes are feasible (L10), the algorithm selects a route with minimum latency and updates (if different) to the current route (L11). Because moving a client to a route with lower latency has a cost and might cause instability, the algorithm only proceeds with changes that will bring a client into the "playable range"; it will not move a client if the best route is not good enough, or if the current route is already good enough. Changes trigger the controller to generate forwarding instructions to update the dataplane to shift traffic from the path with excessive latency to the one with the lowest latency.

5.4 Flow Steering Implementation

The steering of game client traffic flows is achieved by applying segment lists to the dataplane of the ingress node or headend in the network. These segment lists are encoded within an IPv6 extension header called the Segment Routing Header (SRH). The SRH contains the IPv6 addresses of nodes or links in the network that the packet must traverse to reach the desired exit egress node.

Headend placement. In this implementation, any node in the network could serve as the source node responsible for encapsulating packets with segment lists to direct flows to the preferred exit egress node. For instance, the edge nodes at the network's ingress points can be the source node for directing incoming traffic towards a particular exit. Conversely, if the traffic that requires optimisation originates from within the network, the originating node can assume the role of the source node. Alternatively, an intermediate node connected to the edge eBGP or originating node can also function as the source node as long as Overwatch can apply the necessary forwarding instructions and rules to its dataplane.

Headend dataplane using commodity hardware. This implementation used the Linux-based SRv6 node architecture [144] to enable the flow steering functionality. This architecture allows for the coexistence of a local control

logic based on distributed IP routing and a software-defined centralised approach, where the node exposes a southbound API towards a controller. In this context, these SRv6 nodes are utilised as routing devices within the network.

Extending the dataplane southbound API. To enhance the capabilities of these SRv6 nodes, the existing gRPC southbound API defined for these nodes were extended to support latency-aware routing. A protobuf data format is defined to serialise the policies and segment list sent over the gRPC channel to these nodes. The extensions enable the controller to create separate routing tables and define custom policy routing rules to guide specific traffic to the appropriate routing table.

Configuring the dataplane. The controller uses the configuration information, including details about game traffic flows, such as protocol, source, or destination port, to generate instructions for creating distinct routing tables for each traffic flow specified in the configuration. Furthermore, the controller also pushes instructions to implement policies that will mark the traffic flows and direct them to the appropriate routing table using the Linux IP rule and the Netfilter mangle table.

Forwarding instructions for steering flows. For each client (or destination), the controller then adds a route containing the segment lists generated by the latency-aware routing decision process to the relevant routing table. Subsequently, the route for each client is updated periodically during each routing cycle with the appropriate segment list to steer the traffic via another path if the decision changes.

Targeted flow steering. In this implementation, only specific traffic flows are steered through an alternate preferred egress point, as determined by the latency-aware routing algorithm. All other traffic continues to be routed transparently via the BGP best path. This is achieved using basic routing policies and multiple routing tables available in the Linux network stack, as well as other standard network equipment.

Alternative southbound APIs. Importantly, this implementation demonstrates

that latency-aware routing can be realised without needing specialised hardware or niche protocols. Furthermore, it is worth noting that the choice of southbound API used in this implementation is not restricted to gRPC. In [144], various southbound APIs, such as SSH, REST, and NETCONF, which most hardware vendors support, were also considered and implemented.

5.5 Summary

This chapter discussed the implementation of the various subsystems of Overwatch and how they interact to provide latency-aware routing for multiplayer online game traffic. The routing information extraction subsystem aggregates inter-domain routing information for the system. It provides an interface with which Overwatch can exchange routing information with the eBGP peers while adhering to the protocol specification. The latency measurement subsystem periodically conducts RTT measurements to clients using the path control capabilities provided by Overwatch. These measurements are relayed asynchronously to the controller, where they inform the latency-aware routing decisions. The latency-aware routing algorithm uses the measurement and routing data as inputs to make intelligent routing decisions, ensuring that latency-restricted traffic is directed through the path with feasible latency.

The controller subsystem, utilising the Actor model, capitalises on the multi-core processors found in commodity hardware to ensure multiple Actor processes can concurrently and independently execute their tasks. The model also ensures that the processes are not hindered while awaiting responses from other processes. Lastly, the flow steering subsystem introduces a southbound API. The API is responsible for configuring custom routing policies in the Linux dataplane. This, in turn, steers latency-restricted traffic via the path with feasible latency.

Overall, this implementation of the prototype of the Overwatch system uses stable, standardised software and existing hardware functionalities to provide

inter-domain latency-aware routing. It accomplishes this whilst maintaining compatibility and ensuring routing correctness and stability.

Chapter 6

Evaluation

The chapter evaluates the proposed approach, Overwatch, using the prototype implementation described in the previous chapter. We performed a set of experiments running the prototype implementation in the context described in Section 2.3. The results of these experiments confirm our hypothesis that a pragmatic latency-aware routing system can maximise the number of players in a multiplayer game. It does so by exploring the best available paths regardless of the Border Gateway Protocol (BGP) selected path.

This chapter is organised as follows: Section 6.1 justifies the choice of emulation for performing the evaluation experiments. It introduces IPMininet, the chosen emulation testbed and details how the topologies for the motivating scenario in the experiments were configured, along with the necessary modifications required to evaluate Overwatch. Section 6.2 defines the questions we aim to answer and the relevant metrics used. Section 6.3 describes the experiment setup, game server-to-client emulation and various configurations and parameters used in the experiments. Sections 6.4, 6.5, and 6.6 present the experiments and corresponding results. The chapter concludes with a summary and a discussion of the limitations of the evaluation.

6.1 Emulation Testbed

There are three commonly used platforms for networking systems experiments: *simulators*, *physical testbeds*, and *emulators*. *Simulators* replicate the behaviour of a real network without deploying or configuring actual network devices and generating traffic. Rather, events which cause changes in states are used to replicate network and traffic behaviour. Although convenient and easy to "reproduce", there is a lack of functional and traffic realism in simulation experiments. *Physical testbeds* use actual network hardware that is configured, and they carry real traffic for experiments. They are often cost-intensive to build and keep running. They have practical resource limits and may often lack the flexibility to support experiments with custom topologies or forwarding behaviour. This is because they might often be carrying actual traffic. On the other hand, *emulators* can run actual network protocols and applications. They can carry network traffic using virtual hosts that emulate physical devices. There are different types of emulators and emulation, but a *Container-Based Emulation (CBE)* approach is employed for this evaluation.

CBE description. *CBE* is an approach for enabling runnable and reproducible network experiments in an environment of virtual hosts, switches, and links running on a modern multi-core server, using actual application and kernel code with software-emulated network elements. CBE allows actual code to be run on an emulated network using lightweight, OS-level virtualisation techniques. This approach provides topological flexibility, functional realism, timing realism, and traffic realism that can be achieved with network experiments and tests on physical testbeds with ease of replication and low cost. There are, however, concerns about CBE's ability to provide adequate performance isolation for network experiments due to resource provisioning. Nevertheless, it was reported in [72] that the results of some published network experiments could be replicated using CBE, thereby suggesting reproducibility and correctness.

Choice of emulation system. The CBE system used for the evaluation is based on Mininet [91]. It is an open-source framework that leverages

lightweight Linux namespaces to emulate hosts, switches, links, and routers on a single Linux Host. The CBE system, which is IPMininet [18], is a set of Python classes that extends Mininet by providing an API to instantiate Internet Protocol (IP) networks. With IPMininet, network topologies running actual routing protocols, such as OSPF, RIP, and BGP, can be instantiated with the ability to set link weights and configure IP prefixes, routing policies, and configurations.

Topology creation. Running a network experiment on IPMininet requires the user to create a topology. This is done by declaring a Python class used to instantiate and configure hosts, switches, routers, and links between the nodes. The Python class also assigns the hosts and routers to different networks in the topology. The topology is then run by invoking the Python class, which creates an extended Mininet CLI.

Modifying the emulation system. Although the Linux Segment Routing over IPv6 (SRv6) router node was implemented on a physical machine, Overwatch was evaluated using virtual hosts. This necessitated re-creating the functionality of the Linux SRv6 edge router node inside the hosts created in the IPMininet topology. Consequently, the dataplane-manager component, which traditionally receives forwarding instructions and applies them to the dataplane of the ingress/headend node, was adapted and extended to function inside the IPMininet virtual hosts. This component applies the forwarding instructions and configures these headend nodes with the IP rules required to mark flows and steer them into a specific routing table on the hosts. Furthermore, the ExaBGP configuration templates in IPMininet were also modified to facilitate the operation of the sender and receiver modules outlined in Chapter 5.

6.2 Testing Methodology

To evaluate the benefits of Overwatch, a gaming scenario was used in which a game has a set of clients that need to communicate with a server under some

given latency threshold. An IPMininet testbed was used to instantiate the topologies for the gaming scenarios and perform experiments whose setup is described in the next section. The experiments help to answer three questions: (i) Can Overwatch really help? (ii) How well does Overwatch respond to network changes? (iii) What is the impact of varying the main configuration parameters?

Metrics. In order to answer these questions, two metrics were used to highlight the benefits provided by Overwatch and how efficient Overwatch is at providing these benefits. The first metric is the number of players out of the initial set of players that join a game session that Overwatch can keep in the game by preventing them from disconnecting due to excessive latency. The second metric is the degree of "discomfort" experienced by each player through the session. This is represented by the delayed packets each player experiences. The setup of the experiments is described in the section below.

6.3 Experiment Setup

Emulation Environment. As previously mentioned, the experiments were performed using an IPMininet testbed to examine how Overwatch behaves when deployed with existing routing protocols, such as OSPF. The IPMininet testbed was installed and configured on a KVM virtual machine guest with 12 vCPU cores, 24GB of Random Access Memory (RAM), and a 64-bit Ubuntu 18.04 OS. A recent version of the Linux kernel (v4.10 or newer) is required to enable the forwarding of packets embedded with Segment Routing Header (SRH) in the testbed. Resource usage was monitored to ensure no obvious CPU and memory contention effects. However, since it cannot be guaranteed that there was no interference, the results represent the lower bound or worst-case performance.

Experiments Configuration. The experiments are based on two instantiations of the topology shown in Figure 2.1, both representing a gaming scenario

described in Chapter 2. The game network (with Overwatch) has two (or three) egress points via Autonomous Systems (ASes) named A and B (and C). The server connects with up to 300 clients, which are uniformly distributed among 30 different networks (up to 10 players each), all connected to AS X. In some experiments, the BGP choice of egress point in the game network or the end-to-end latency of clients is changed to see how Overwatch reacts.

The acceptable latency threshold is defined as 60ms. Each player is configured initially with some end-to-end average latency around the threshold. Considering all players, the latency distribution differed according to the topology but, in both cases, was arbitrarily chosen to highlight the behaviour. In the first one, the (average) latencies of players are uniformly distributed in the range 54.5 – 69.5ms when via A and 50 – 65ms when via B, with half the players in each set. In the second topology, there are three egress points, and the 30 networks (with 10 clients each) are split into three groups, respectively via A, B or C. All the players going via A have an *initial* average latency equal to 50.0ms; for B and C, they are 45.0ms and 40.0ms, respectively; such values are reconfigured during the experiment depending on the purpose. Simple checks with ping and traceroute were used to validate the latency values and paths under different BGP configurations.

Game emulation. There is no generic template for how latency-sensitive online games behave, but the model is based on the traffic characterisation work in [51]. Each player sends small UDP packets with state to the server. Typically, a server will collect packets from all players and periodically "broadcast" consolidated information to players. However, in this case, the server immediately responds to allow computing metrics on the player side. A UDP echo server application was implemented in Python to emulate the game server. It receives packets from the players and echoes them back to the players, allowing the latency calculations and disconnection logic to be implemented on the player side. A UDP application in Python was also implemented to emulate a game client player and the disconnection logic. The application sends packets to the

server and calculates the Round-Trip Time (RTT) by getting the difference between the time when the packets were sent and when they were received back from the server. The application also logs the RTT and disconnects when the disconnection threshold conditions are met.

Disconnection threshold. The players are configured to disconnect from the session whenever one of these conditions occurs: (i) over 25% of the measurements in the last 180 seconds exceeded the maximum latency threshold for the game, or (ii) 100% of the measurements have exceeded the threshold in the last 30 seconds. The criteria for disconnection is based on the findings in [65, 25], reflecting the behaviour of different gaming platforms when latency increases. Players join the game with deterministically decreasing inter-arrival times, following an exponential decay from an initial inter-arrival time of 200s to a minimum of 1s. To keep the experiments tractable, each player's session is set to a fixed duration of 10 minutes, which can be terminated early if the player experiences latency above the maximum acceptable threshold. While actual game session lengths vary by genre, with top action games averaging 9 minutes per session [66], this fixed duration was chosen to standardise the experiment parameters.

Overwatch. Unless otherwise noted, the experiments were run using a routing period T of 20 seconds and a measurement period M of 1 second. In the algorithm, K was set to 4, having experimented with a range of values. The impact of parameters is discussed in the sensitivity analysis section. In the experiments, the prototyped latency measurement subsystem used a set of additional IP addresses associated with the available egress points. It utilised a distinct IP address and port combination for each available path via the egress points, employing these pairs to exchange probe packets with players via respective paths. In the ingress/headend router nodes, firewall rules and IP rule policies generated by Overwatch are used to parse and classify the packets that require latency-aware routing so that they can be steered accordingly (to a specific routing table dedicated to that egress point). Overwatch generates

these rules and policies based on the specifications defined for game traffic flows and information in the configuration. Overwatch then applies these policies to these nodes via the southbound API mechanism described in Section 5.4.

6.4 Can Overwatch Really Help?

This section seeks the answer to the question: How many players can Overwatch "bring to the game session" and keep them playing? The answer depends on the number of players with excessive latency and how many of these players could be connected via a path with feasible latency. If all players are connected with feasible latencies, Overwatch will not provide any benefit. If the players with excessive latency do not have any paths with feasible latency, then again, Overwatch cannot help them. To avoid exaggerating the benefits, both these cases were included in the evaluation and in equal proportions (this split was arbitrary). This was implemented by uniformly distributing clients in latency ranges.

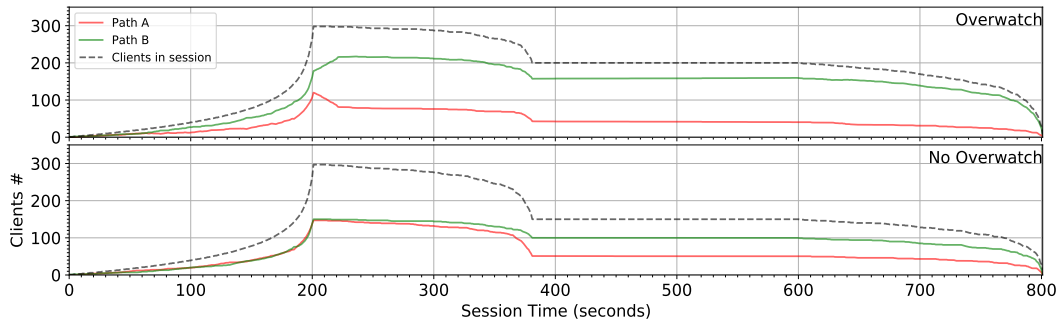
Experiment description. The experiments should confirm the benefits of Overwatch and, when not, why. Two sets of experiments were performed to answer this question with the topology with two paths, via A and B. In this topology, 300 players join the game and are allocated to paths by BGP, half on each path via A and B. Given the latency on both paths in the topology, if BGP selects path B for all game players, most players will stay in the game, but a few will still disconnect soon after arrival. In contrast, if BGP selects path A for all, most players will disconnect soon after arrival, but a few will stay in the game. The first set of experiments is performed without Overwatch, while the second set is performed with Overwatch.

Results. Figure 6.1 illustrates the impact of Overwatch on the "survival" of players until the end of a session, contrasting networks with Overwatch (top) and without (bottom). More specifically, Figure 6.1a depicts the average number of players in the session through time, the allocation of players to paths

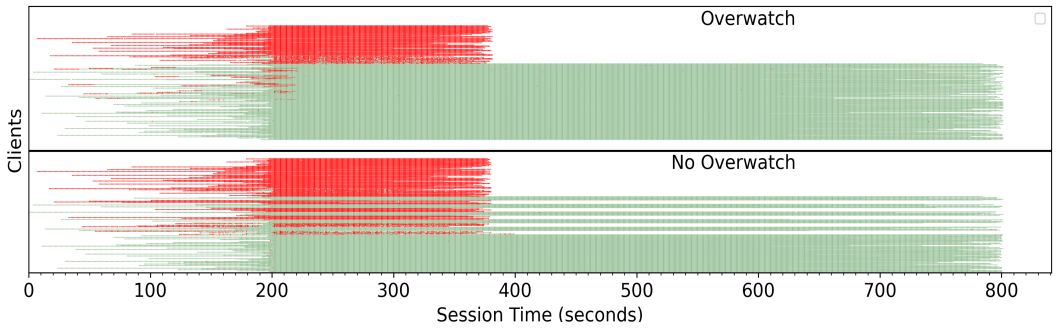
initially by BGP (half on each path), and the subsequent shift to another path when Overwatch intervenes. Figure 6.1b, on its turn, shows the level of "discomfort" experienced by each player through time, as represented by the delayed packet events. Events coloured in red in the second plot indicate that the latency of the packets exchanged between the player and server exceeded the threshold. In contrast, green-coloured events show that the latency of the packets exchanged remained below the threshold. Each line in the plot represents the events for an individual player, organised in ascending order of configured path latency, from bottom to top in each half of the plot.

In Figure 6.1a, it can be seen that up to 200s, 300 players join and are allocated to paths by BGP, half on each path, A and B, but Overwatch detects that the latency is excessive for some 20 clients and steers them from A to B. Up to 380s, the disconnection timeout gradually kicks in for the players with "excessive" latency, and these players disconnect. Because of how latencies were configured (Section 6.3), disconnections happen on *both* paths and regardless of Overwatch, but in a smaller proportion with Overwatch. When comparing the dashed lines at the top and bottom, it can be seen that Overwatch provides a 33% decrease in the number of players with excessive latency, "bringing in" 50 additional players.

Figure 6.1b shows that without Overwatch (bottom), 150 players experience frequent delays above the threshold, as indicated by the consistently red events. After enduring delay above the game threshold for 180s, these 150 players eventually disconnect from the server. In contrast, with Overwatch, some players on the path via A benefit from lower latencies via B. After collecting latency measurements on both paths throughout the routing period (20s), Overwatch intervenes and shifts these players to the path via AS B, limiting their delays.



(a) Clients in game session



(b) Excessive delay events

Figure 6.1: The impact of Overwatch on player success when latency is excessive for some players via BGP's best path. The server has 300 players connected, uniformly distributed among the two paths.

6.5 How Well Does Overwatch Respond to Network Changes?

How many players can Overwatch "keep" in the game when network events that trigger changes occur? Here, the aim was to illustrate how well Overwatch can detect network changes and respond accordingly. This question was addressed using two distinct scenarios: (i) topology with two paths, via A and B, when a change prompts BGP to shift from a below-threshold path to an above-threshold one; and (ii) topology with three paths, when multiple latency changes occur. In both scenarios, the aim is to illustrate that Overwatch can effectively detect these network changes and respond accordingly. In the first scenario, Overwatch's benefit should be shielding players from the adverse effects of a BGP change. In the second scenario, Overwatch's response should involve moving players impacted by latency increases on the current path to an alternate path offering under-threshold latencies for those players. The same metrics were used as in Section 6.4 to quantify the benefit provided by Overwatch in the event of a change in network conditions.

6.5.1 BGP Changes Egress Point

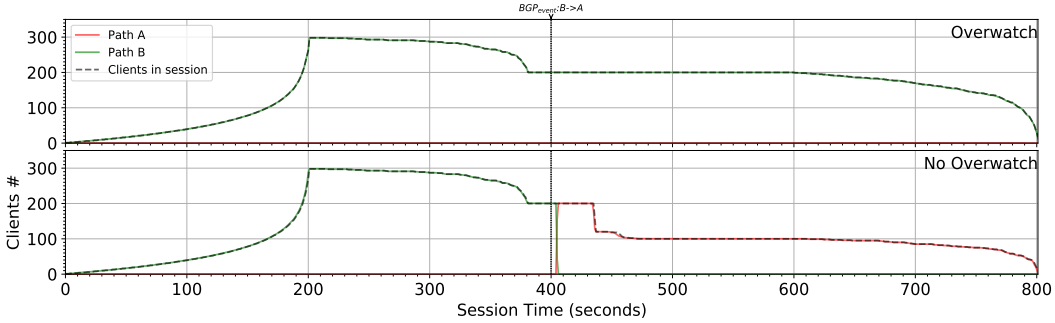
Experiments were performed with and without Overwatch to evaluate the effect of Overwatch on the "survival" of players to the end of a session when an event caused BGP to change its choice of path during the game session. The topology with two paths via A and B was used in these experiments. In these experiments, all players initially arrive at the game via a single path instead of being distributed across paths A and B. Figure 6.2a shows the "survival" of players on average when an event causes BGP to change its path choice. Initially (up to 200s), all players arrive at the game being routed via B, the egress point with lower latency. However, some players still have excessive end-to-end latency and cannot be helped by Overwatch (they are already in B). This illustrates again that there are cases in which Overwatch will not be able

to help. These players disconnect after 180s, and the corresponding drop is seen. At 400s, a BGP event occurs, changing the preferred path from egress B to A, the egress point with higher latency. It takes 5s for the routers to converge and move the clients' prefixes from the path via B to A. Without Overwatch, the change from B to A leads to increased latency, causing 100 more players to disconnect, leaving only 100 in the game. In contrast, Overwatch keeps players on the path via B despite the BGP change. As a result, 100 additional players (200 in total) can stay in the game.

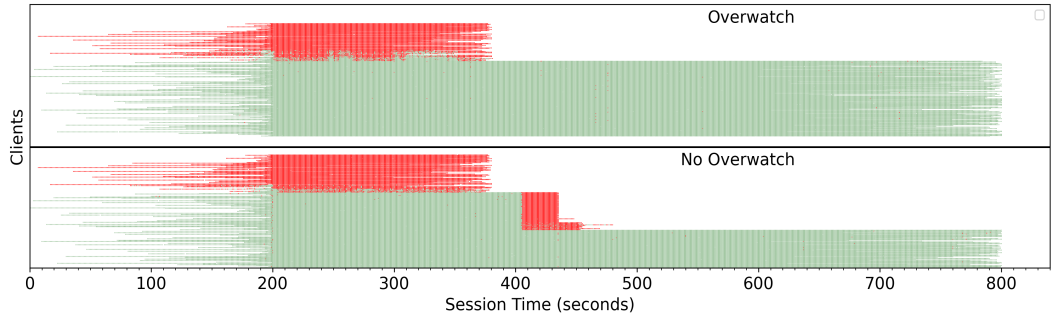
The delayed event plot in Figure 6.2b illustrates the resiliency and protection that Overwatch provides to players against the impact of the BGP change. In the lower half of the plot, the delayed events experienced by the players without Overwatch were observed. Immediately after BGP switches players from path B to A, some players start experiencing delayed events and eventually disconnect, leaving fewer players in the session till the end. In the upper half of the plot, the delayed events experienced by the players with Overwatch are seen. It becomes evident that these players do not suffer any adverse effects from the BGP change, as Overwatch keeps the players on path B.

The plot in Figure 6.2c further illustrates the effectiveness of Overwatch in mitigating the impact of BGP changes on delays experienced by players. The solid and dashed lines represent the average count of on-time and delayed packets for all clients through time. Initially, as players join, there is a rise in delayed packets for both situations – with and without Overwatch, caused by players with "excessive" latency. This trend reverses as these players gradually disconnect. Following a BGP change that shifts players from path B to A, a significant increase in delayed packets and a corresponding decrease in on-time packets are observed for the packets without Overwatch. This observed behaviour indicates that 100 players are experiencing "excessive" latency. These affected players eventually disconnect, leaving only 100 active until the end. Contrastingly, Overwatch maintains players on path B despite the BGP change, thereby shielding the players from the negative impact of the

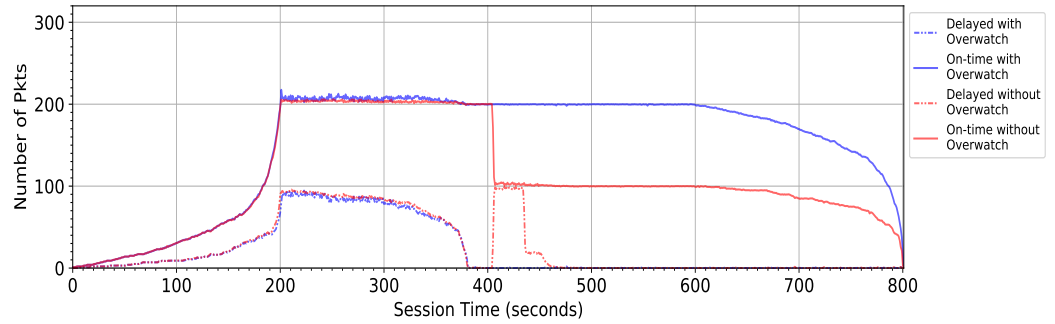
change, as evidenced by the stable count of on-time packets.



(a) Clients in session with and without Overwatch.



(b) Delayed packets with and without Overwatch.



(c) Number of delayed and on-time packets.

Figure 6.2: The impact of Overwatch on player success when BGP updates path choice, preferring A over B. The server has 300 players, all on the same path.

6.5.2 Latency Changes (Three Egress Points)

To further validate Overwatch's effectiveness in responding to multiple network changes with multiple "feasible" path choices, the second topology with three paths, described in Section 6.3, is now used. The 300 players are evenly distributed across the paths through A, B and C. Figure 6.3a shows the average number of players in the game over time with (top) and without

Overwatch (bottom).

Players arrive initially (up to 200s), and their latency remains below the threshold regardless of arriving via A, B, and C. Four latency changes from 350s occur (one at a time), spaced by 70s, rendering one or more paths infeasible. Without Overwatch (bottom), a latency increase on the path via A leads to delay-induced disconnections, reducing the player count to 200. Although latency via path A recovers 70s later, a subsequent latency increase on the paths through B and C at 490s results in the disconnection of the remaining players. By 560s, while the latencies on the paths through B and C "recover", no players remained in the game.

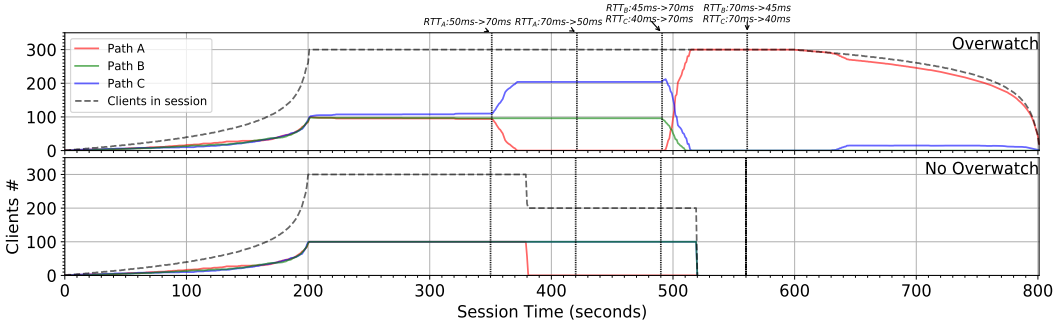
This starkly contrasts the behaviour when Overwatch is running (top of Figure 6.3a). We observe that players' game continuity is maintained until the end under the same conditions. Upon the initial latency rise on the path via A, Overwatch gradually shifts the affected players to the path via C and refrains from reverting them post-recovery 70s later. At 490s, with the latencies via the paths through B and C increasing beyond the threshold, Overwatch reacts and transitions all players to the path through A, thus preventing disconnections. Subsequent latency normalisation on the paths through B and C 70s later prompts a minor redistribution of players to C because their high RTT variation on the current path renders it infeasible (according to the algorithm). However, all players remain in the game till the end.

The delayed events plot in Figure 6.3b illustrates how Overwatch responds to these changes and its impact on the delayed events experienced by the players. Without Overwatch, the players encounter excessive delay events and subsequently disconnect. With Overwatch, players are shifted to the best path available at the end of their routing periods. Figure 6.3c delineates the adaptive response of Overwatch to network changes and their impact on the delay experienced by the packets exchanged between the players and the server. Without Overwatch, the initial network event at 350s results in a

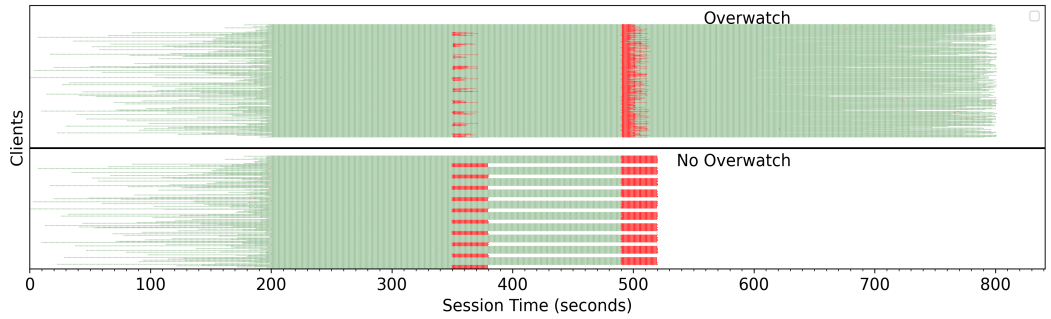
discernible dip in on-time packets and a surge in delayed packets, leading to player disconnections within 30s. In contrast, with Overwatch, there is a temporary rise in delayed packets, but players are swiftly reallocated to a path with feasible latency, normalising delays. The subsequent event 70s later has no significant delay impact on players in both scenarios, unlike the third event at 490s, where the delay increase on paths B and C precipitates increased "delayed" packet counts and disconnections without Overwatch. Conversely, with Overwatch active, the system reallocates all players experiencing excessive delays to a better path, returning the delayed packet count to negligible values. The final event showcases Overwatch's efficacy, as no fluctuation in on-time packet count occurs, ensuring all players remain connected until the end.

6.6 Sensitivity Analysis

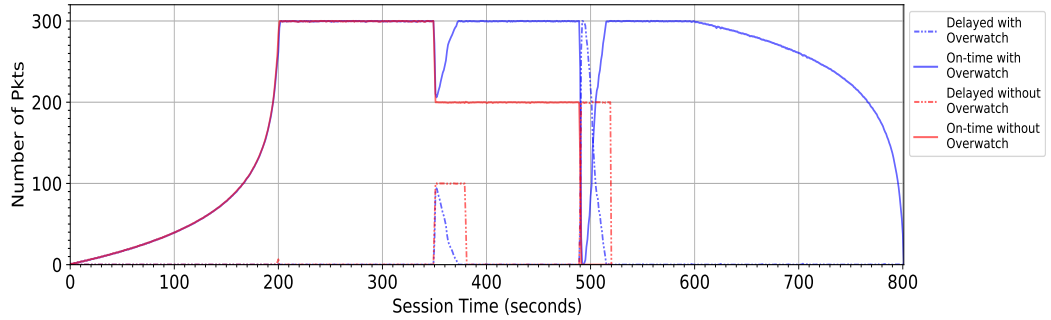
Parameter analysis. Overwatch has two key parameters. The routing period, T, determines how frequently the control algorithm runs, potentially leading to updates of the selected paths. The measurement period, M, defines how often measurement packets are sent to the clients on the available paths. The optimal combination of these parameters depends on both application requirements (e.g., the disconnection threshold) and network conditions. There is an apparent trade-off between responsiveness/accuracy and overheads (control messages, processing). A shorter T gives the control algorithm more opportunities to run and update its choice of paths for clients, so results are expected to improve with lower values but with more considerable overhead. A routing period exceeding the disconnection threshold (30s) implies that the control algorithm might only update the client's path choice after the opportunity to prevent disconnection has elapsed, and this issue progressively worsens as the routing period further surpasses the disconnection threshold. With the measurement period, a shorter M provides more accuracy in measuring latency but at the cost of extra control packets.



(a) Without Overwatch, multiple players disconnect after changes.



(b) Delayed packets with and without Overwatch.



(c) Number of delayed and on-time packets.

Figure 6.3: The impact of Overwatch on player success when multiple latency changes happen.

This section shows the impact of these two parameters within the context of the motivating scenario.

6.6.1 Sensitivity Analysis Experiment Setup

The same scenario discussed in Section 6.5.2 was employed, with three egress points with multiple delay changes. Experiments were run for combinations of T and M, with similar metrics: the number of players who keep playing and the maximum number of consecutive delay packets for players on average. Without

Overwatch, clients would inevitably face disconnection in such circumstances. Therefore, the goal is to evaluate the efficacy of various routing periods (and measurement periods) in retaining clients in the game under those conditions. This assessment offers guidance on selecting the appropriate routing periods within a specified disconnection threshold.

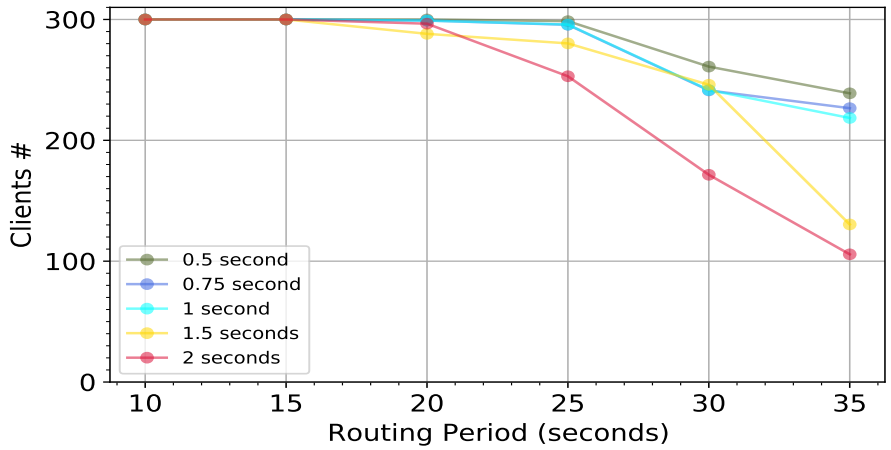
6.6.2 Impact on Saved Players

Using the default measurement period, 1s, T was varied between 10 and 35s. The results are shown in Figure 6.4a. Overall, as expected, increasing the routing period T initially does not have any effect on saving clients, but eventually, performance deteriorates. Decreasing T below 20s provides no benefits, while increasing above 20s gradually has a negative impact. The optimal choice of T depends on M, but for larger values of T, differences in M can significantly affect the performance. For example, when T=30s and M=2s, Overwatch can keep 175 clients compared to the 258 saved when T=30s and M=0.5s. Comparing the five curves confirms that M has a substantial impact: Overwatch depends on having fresh latency measurements when network conditions change abruptly multiple times.

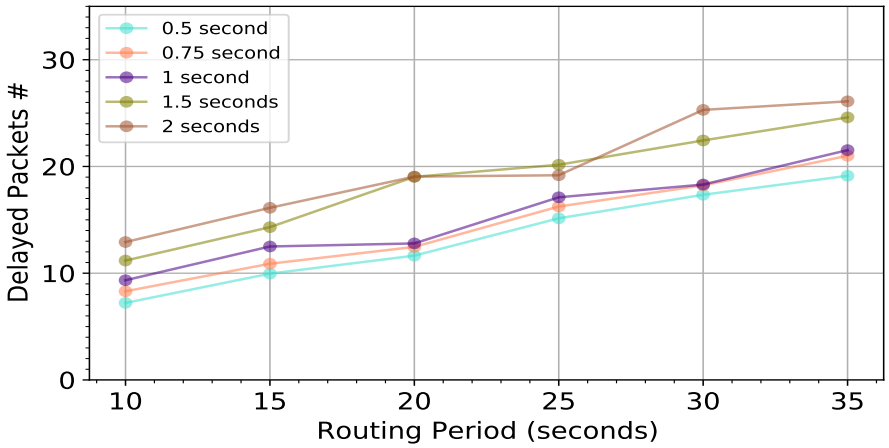
6.6.3 Impact on Delayed Packets

The effects of increasing the routing and measurement periods are also noticed in the consecutive delayed packets in Figure 6.4b. It shows that the consecutive delayed packets remain low for small values of T and M, with conditions deteriorating as the measurement and routing periods increase. Depending on the tolerance of the application to short periods of excessive latency, a combination of a longer M and a shorter T can be effective, such as 2s and 10s for M and T. Combinations like this would reduce measurement frequency while providing the control algorithm with enough opportunities to run and update its choice of paths for clients. To illustrate, when T is 10s, Figure 6.4b indicates that the number of consecutive delayed packets increases by 5 from

the shortest to the longest M, but connectivity is maintained for all clients (Figure 6.4a). Ultimately, we observed that shorter routing periods, well below the application threshold, minimise the consecutive delayed packets across various measurement periods. Therefore, to minimise the level of "discomfort" experienced by players, the optimal strategy is selecting a routing period significantly under the application threshold regardless of measurement periods.



(a) Clients saved.



(b) Maximum consecutive delayed packets.

Figure 6.4: Clients saved and delayed packets for the different combinations of T & M.

6.7 Result Discussion

This chapter sought to provide evidence about the goodness and feasibility of Overwatch by attempting to answer three questions: (i) Can a pragmatic

latency-aware routing solution like Overwatch maximise the number of players that communicate with a game with *feasible* latencies by dynamically adjusting the server-to-clients paths? (ii) How well does Overwatch achieve this goal under different network conditions? (iii) How does varying the main configuration parameters affect Overwatch's ability to achieve the stated goal of maximising clients that communicate with the game under a predefined threshold?

Overwatch's goodness. Based on the experiments and results collected from the experiments, we demonstrated in an emulation environment that Overwatch can effectively maximise the number of players connected to a game server when there is excessive latency. When players arrive at the game server via a path selected by BGP with excessive latency, and there is a better, viable path for them, Overwatch can intervene and shift these players to a path with acceptable latencies, thus preserving them in the game. Given the slow convergence of BGP after a change which is disruptive to real-time network applications, we demonstrated that Overwatch could shield and safeguard game traffic flow from changes in network conditions caused by a BGP change. Apart from control-plane induced changes, we showed that Overwatch could respond to network conditions that are not routing-induced by shifting game players to more viable paths in response to those changes. We evaluated the ability of Overwatch to make these decisions when multiple feasible path choices are available under different conditions. The results confirmed that Overwatch judiciously changes player paths only when there is a distinct advantage, avoiding unnecessary shifts.

Impact of operating parameters. Finally, we assessed the two crucial parameters of Overwatch to determine their impact on its effectiveness given a specified disconnection threshold. We found that because Overwatch uses a weighted average of the RTT in the algorithm, a shorter measurement period results in more precise and efficient decisions by Overwatch because it collects more measurement samples within a given period, thus providing a more accurate view of network conditions. However, with a longer measurement

period, fewer measurement samples are collected within the same given period, thus resulting in Overwatch making a decision later than needed. Depending on the threshold, Overwatch's late reaction might result in clients disconnecting because the threshold has been violated. For applications with more tolerant thresholds, when using a more extended measurement period, it might be more efficient to use shorter routing period durations to account for this and get Overwatch to evaluate the paths more often.

The sensitivity analysis experiments also showed that routing periods up to 83% of the disconnection threshold could be used, resulting in the same efficiency level for a measurement period up to 0.75 seconds. This reduces to 66% when the measurement period increases to 1 second and progressively worsens.

6.7.1 Evaluation Assumptions and Limitations

Several assumptions were made regarding the game and the network configuration in developing and evaluating Overwatch. For instance, one assumption was configuring the players to disconnect when their latencies to the server exceeded a maximum latency threshold. While it is widely held that game providers have no incentive to disconnect players due to excessive latency, users typically disconnect because of poor gaming experience. In the game emulation used for the evaluation, the players were configured to disconnect due to excessive latency. This setup was crucial for quantitatively assessing the impact of excessive latency. This insight was validated by a network engineer for one of the largest gaming company providers. The engineer also indicated that game servers handle far more than the 300 clients used in the tests. However, due to the limitations of the emulation environment, it was not feasible to scale the evaluation of Overwatch to thousands of clients.

Chapter 7

Pragmatic PAR for Additional Applications

This thesis addressed the problem of providing *feasible* latency to players in multiplayer online games without significant changes to the Internet's core. A pragmatic, latency-aware routing system using Border Gateway Protocol (BGP) Egress Peer Engineering (EPE) was introduced to achieve this. This system is designed to be compatible with existing Internet protocols and infrastructure, thus removing the need for specialised hardware or developing new protocols.

Overwatch successfully ensures latency-aware routing by continuously monitoring real-time traffic and performance. It maintains application operations below a predetermined latency threshold, provided alternative routes are available. This capability suggests that Overwatch's potential applications extend well beyond its current use, offering benefits to many networks and applications.

Section 7.1 explores the potential of extending Overwatch to function in network environments other than game provider networks. It also details the requirements for implementing Overwatch in these environments. Additionally, there is an examination of how Overwatch could be adapted to accommodate diverse algorithms and performance goals, including bandwidth optimisation, congestion management, and secure routing paths for different applications.

Finally, considering Overwatch's development on Linux-based router nodes, there is an examination into how it can be integrated with whitebox network appliances to facilitate latency-aware routing in Section 7.3.

7.1 Application to Other Networks

We believe Overwatch, initially implemented within a gaming provider network for latency-aware routing of game traffic, holds potential for broader applications in other network environments. We discuss its viability in two distinct network environments where it could be leveraged to provide latency-aware routing.

Transit networks. First, Overwatch can be deployed in transit networks to offer latency-aware routing as a value-added service. About 46% (from the analysis in Section 2.3.1) of stub networks with only one upstream will not benefit from running Overwatch in their networks. However, approximately 77% of Tier-2 and Tier-3 transit providers have two or more upstream providers, not including their peers. Therefore, for those stub networks with a single transit upstream, their respective transit providers could integrate Overwatch to provide both transit services and latency-aware routing.

In this scenario, the transit provider could utilise Overwatch for latency-aware routing for all customer-originating traffic flows or limit it to specific, predetermined destinations. The eBGP device at the transit provider's end, which peers with the customer, serves as the ingress headend node. This node would be configured to redirect flows from a less optimal upstream or egress point to a more favourable one. The *flow steering* subsystem is deployed at the network's edges, particularly at points where traffic enters the network. This arrangement maintains a separation between the core and edge of the transit provider network, thereby reducing its complexity.

Measurement in a transit network. In a transit network scenario, the measurement subsystem requires re-implementation to address two pivotal challenges in latency measurement. The first challenge involves identifying

flows that require latency-aware routing with minimal cooperation between the transit and customer networks. In contrast, the second pertains to the scalable performance assessment across a substantially more extensive set of destinations via the available paths.

Flow identification. Transit providers may employ deep-packet inspection (DPI) appliances to identify these specific flows. However, DPI appliances may be inadequate for reliable, stateful flow identification and matching for connection-less services, like online games, which typically use UDP. In that case, alternative techniques like automatic signature extraction, as exemplified in [96], could be effectively utilised.

Enabling large-scale probing. Commodity hardware within the transit provider's network can be used for latency measurements for a wide range of destinations via multiple upstreams. After identifying flows that require latency-aware routing, their respective remote destination IP addresses are relayed to the Overwatch controller subsystem. This controller configures specific rules to guide the measurement traffic the measurement subsystem generates through the available paths. Establishing a high-performance latency measurement system is possible even without control over the end hosts in the flow exchange. Efficient tools like gufo-ping [69], which offers a Python API for sophisticated raw sockets manipulation, can be utilised. It enables the simultaneous probing of over 100,000 hosts. However, this method has limitations; ping packets may be deprioritised or dropped in network routers, potentially affecting the accuracy of the reported Round-Trip Time (RTT) in reflecting the end-to-end latency. Nonetheless, these measurements should provide a sufficiently accurate representation of path performance for Overwatch to make informed routing decisions compared to BGP.

Wide-Area Network (WAN) and enterprise network application. Overwatch applies not only to transit networks but can also benefit large enterprise networks with multiple WAN connections between their sites. These networks often deploy diverse provider connections to ensure high availability

and swift failover between their sites. Overwatch can enable these networks to exploit all their WAN connections simultaneously. The visibility into the performance of all connections can then be leveraged by Overwatch to dynamically reroute specific flows via the optimal connection.

7.2 Extending Overwatch

Currently, Overwatch ensures that the latency between a group of clients and a game server remains within a predefined threshold, provided alternate paths are available. This section delves into the possibilities of extending Overwatch to meet various objectives, outlining the requirements and implementation strategies.

Ease of extension and adaptation. One of Overwatch's key design features is its extensibility, stemming from its reliance on stable, standardised software, existing hardware functionalities, and deployment on commodity hardware. This foundation simplifies the system's extension, avoiding added complexities. Overwatch is developed in Python, utilising the Actor model. As described in Section 5.3, each process within the controller subsystem functions as an independent, concurrent entity that asynchronously exchanges messages with each other. This modular structure allows for the incremental integration of new features and components, maintaining system stability and minimising the need for extensive modifications to existing components. Although the current implementation of Overwatch operates all processes on a single node, the Actor model's inherent modularity and flexibility enable the distribution of these processes across multiple nodes. They can communicate asynchronously using various inter-process communication methods, enhancing the system's adaptability and scalability.

Having established Overwatch's extensibility and adaptability, three key areas of potential expansion are considered: (i) enhancing the functionality of the latency-aware routing algorithm for diverse applications, (ii) utilising

Overwatch for congestion and capacity management at egress links, and (iii) implementing secure path routing.

Extending the algorithm for latency-aware routing. Extending the latency-aware routing algorithm is relatively straightforward. As detailed in Section 5.3, this algorithm operates as a process within the controller subsystem and is developed using Python. The algorithm, encapsulated as a class within the controller code, can be easily modified to alter its behaviour. For example, the algorithm could be adjusted to consistently direct flows along the lowest latency path or proactively move clients to better paths when latency rises, even if it remains below a certain threshold. Notably, such extensions would not necessitate changes to other subsystems in Overwatch, given the existing support from the latency measurement subsystem.

Extending Overwatch for other performance objectives. Overwatch offers potential for capacity and congestion-aware routing by directing specific flows through less congested egress links. This feature is particularly beneficial for applications that are sensitive to congestion. Implementing this capability would involve developing a congestion-aware routing algorithm to manage flow direction and avoid congestion. A new measurement subsystem must also be created to assess congestion levels at egress links. This could be done passively by analysing traffic flow data gathered or streamed through services like NetFlow [33] and IPFIX [5].

Enforcing security policies for inter-domain routing. Overwatch's capabilities can be expanded to facilitate secure path routing [29] or enforce the use of validated paths [15] for traffic directed towards specific destination prefixes. With Overwatch's ability to steer flows to any chosen egress independently, bypassing BGP and underlying Interior Gateway Protocol (IGP) policies, it can be employed to enforce security policies. These policies might involve avoiding certain upstreams or Autonomous Systems (ASes) because of security considerations. A conceivable scenario involves a security application analysing Overwatch's routing tables, which compile multiple routes from various eBGP

sources. This application could then establish predefined or dynamically updated rules to create whitelists of routes. Based on these whitelists, it would generate forwarding instructions to guide specific flows along the approved path to a prefix.

7.3 Interoperable Flow Steering

In the Overwatch implementation, a Linux-SRv6 node was modified as the BGP EPE router, responsible for steering the flows. The southbound API in the Linux-SRv6 node was updated and modified, enabling Overwatch to alter the dataplane state on these nodes. This sections discusses how flow steering can be achieved using white-box appliances.

A significant obstacle in implementing flow steering arises with network equipment that utilises closed and proprietary Network Operating Systems (NOS) provided by manufacturers. However, more open and customisable NOS options exist, facilitating the integration of local control logic based on distributed IP routing and a centralised, software-defined approach. Notable examples included Service Router (SR) Linux [126] from Nokia and Cumulus Linux [108] from NVIDIA Cumulus Networks. These systems are stable, supportive of open-source networking and offer well-defined interfaces for control applications to interact with their dataplane.

These NOS can be installed on bare-metal devices and white-box appliances, configured to use a chosen southbound API for receiving Overwatch's forwarding instructions and directing flows to preferred egress nodes. Devices equipped with this NOS would serve as the headend ingress node, while other devices in the network operate with Segment Routing over IPv6 (SRv6) extensions for the IGP running on these devices. As such, minimal upgrades and changes to existing equipment running closed NOS would be required to deploy Overwatch.

Chapter 8

Conclusion

8.1 Summary of Thesis

This thesis presented Overwatch, a practical solution to provide inter-domain latency-aware routing for client-server multiplayer game applications. Latency below a specific threshold is essential for the performance of these applications because it affects the number of clients that can connect to a server and their gameplay experience on the server. However, the best path concerning latency for these clients may differ from the default path selected. This difference arises because Border Gateway Protocol (BGP), the de facto inter-domain routing protocol, has inherent limitations such as constrained Traffic Engineering (TE) capabilities, a lack of multipath routing ability and an inability to dynamically respond to changes in network conditions. These limitations render BGP incapable of providing the latency-aware routing required for these interactive applications out of the box. Prior works that have tried to address these challenges have not gained widespread adoption because they either require significant changes to the core of the Internet (by replacing BGP – an approach that is not feasible) or specialised hardware equipment and cooperation between networks.

An alternative approach to latency-aware routing. Overwatch offers a different approach to address the problem. It is a routing control system

that utilises Egress Peer Engineering (EPE) to steer flows to a different egress independently and transparently of the underlying Interior Gateway Protocol (IGP) and BGP policies in the network. This system uses commodity hardware and focuses on steering traffic via the most feasible egress based on real-time latency measurements. Importantly, Overwatch achieves its objectives whilst ensuring compatibility with existing BGP frameworks and maintaining route stability and safety.

Achieving pragmatic explicit routing. Latency-aware routing for specific flows requires fine-grained forwarding instruction to achieve it. In implementing Overwatch, we utilised a Segment Routing over IPv6 (SRv6) dataplane that receives forwarding instructions from the controller subsystem via a gRPC southbound API channel. In previous works, this was achieved using specialised hardware and network fabric. This approach, however, enabled flow steering instructions to be programmatically applied using existing protocols and systems.

Identifying conditions for latency-aware routing. We defined a motivating scenario involving multiplayer games to demonstrate the application of latency-aware routing. The scenario established that the conditions in which Overwatch can be beneficial exist by verifying the presence of multiple transit providers in different networks. We performed latency measurements on three major game service networks to identify the percentage of networks that could benefit from Overwatch within a limited geographic area. Additionally, we used the latency data gathered from measurements between networks in a specific geographic region to the three game servers to guide the evaluation.

Evaluating Overwatch. We performed an evaluation of the Overwatch implementation in an emulation testbed to assess the benefits of Overwatch and the impacts of its parameters on its effectiveness. We aimed to answer three research questions through experiments performed in the evaluation section, thereby testing our hypothesis. The questions are: (1) Can Overwatch ensure that the maximum number of players communicate with the server

under a predefined latency threshold when latency on BGP's default path is excessive for game flows? (2) How effectively does Overwatch achieve this objective amidst changing network conditions? (3) What is the impact of varying the main operating parameters of Overwatch on its effectiveness?

Evaluation findings. The results of the experiments indicate that Overwatch reduces the number of clients experiencing excessive latency when BGP's choice of path is sub-optimal. In scenarios where BGP shifts from a below-threshold to an above-threshold path for all clients, the experiments show that Overwatch keeps twice the number of players in the game compared to BGP. In addition to improving the number of clients preserved, Overwatch shields and isolates the clients' flows from the disruptive effects of BGP convergence when a BGP change occurs. The experiments demonstrated that Overwatch can accurately respond to multiple latency-related network events. It makes the right decisions when multiple feasible paths are available, confirming that the control algorithm does not cause instability by shifting players to another path without clear benefits. Furthermore, our evaluation illustrates that the measurement period significantly impacts Overwatch's performance and effectiveness. Due to the weighted average applied in the Round-Trip Time (RTT) measurements, we observed that a shorter measurement period results in a more precise and efficient decision by Overwatch. However, as the routing period approaches the disconnection threshold, Overwatch becomes less efficient.

Additional application of Overwatch. This work also discusses the potential of implementing Overwatch in different networks and for other applications. Then, it outlined the necessary extensions and modifications for adapting Overwatch to these new contexts. This work also highlighted the practical considerations for implementing flow steering in real-world networks using white-box equipment.

8.2 Future Work

While Overwatch successfully achieves its primary objective, additional opportunities for improvement remain. Currently, the prototype implementation does not factor in scenarios with multiple paths available through different transit providers or peers at an egress link. Also, the system's network resource utilisation could be further improved. The routing control algorithm, as it stands, does not dynamically adjust measurement and routing period intervals in response to changing network conditions. Incorporating such adaptability would allow the system to tailor its network probing activities to the current demands.

Moreover, Overwatch's scope is confined to well-connected stub networks and does not account for transit costs, even when an egress point via a peer is as feasible as one via a transit. The architecture also needs enhancements in terms of robustness, with better protection and quicker failover capabilities in case of controller or dataplane issues or during maintenance.

The evaluation of Overwatch was in an emulated environment, limiting the ability to test it with thousands of clients. The flow steering subsystem was implemented using SRv6. In the future, there is potential to test Overwatch in real network environments with actual application traffic and to examine its scalability with a more extensive client base. Investigating adaptive sampling methods could further optimise network resource management. Future research should also consider Overwatch's application in transit networks, including how it manages transit costs. Additionally, exploring the utilisation of multiple paths from different providers at a single egress link and adapting this for various dataplanes would be valuable. Lastly, enhancing the architecture with a collaborative, multi-controller setup could significantly improve resilience and speed up failure recovery.

Bibliography

[1] Maha Abdallah, Carsten Griwodz, Kuan-Ta Chen, Gwendal Simon, Pin-Chun Wang, and Cheng-Hsin Hsu. 2018. Delay-sensitive video computing in the cloud: a survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14, 3s, Article 54, (June 2018), 29 pages. ISSN: 1551-6857. DOI: 10.1145/3212804. https://doi.org/10.1145/3212804.

[2] Ernest Adams. 2014. *Fundamentals of Game Design*. (3rd edition). New Riders Publishing, USA. ISBN: 0321929675.

[3] 2022. AFRINIC extended delagation statistics, 2022-09-23. https://ftp.afrinic.net/pub/stats/afrinic/2022/delegated-afrinic-extended-20220923. (2022).

[4] Gul Agha. 1986. *Actors: a model of concurrent computation in distributed systems*. MIT press.

[5] Paul Aitken, Benoît Claise, and Brian Trammell. 2013. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011. (September 2013). DOI: 10.17487/RFC7011. https://www.rfc-editor.org/info/rfc7011.

[6] 2022. APNIC extended delagation statistics, 2022-09-23. https://ftp.apnic.net/stats/apnic/2022/delegated-apnic-extended-20220923.gz. (2022).

[7] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. 2021. Performance-driven internet path selection. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)* (SOSR '21). Association for

Computing Machinery, Virtual Event, USA, 41–53. ISBN: 9781450390842. DOI: 10.1145/3482898.3483366. https://doi.org/10.1145/3482898.3483366.

[8] Thomas H. Apperley. 2006. Genre and game studies: toward a critical approach to video game genres. *Simulation & Gaming*, 37, 1, 6–23. DOI: 10.1177/1046878105282278. eprint: https://doi.org/10.1177/1046878105282278. https://doi.org/10.1177/1046878105282278.

[9] 2022. ARIN extended delagation statistics, 2022-09-23. https://ftp.arin.net/pub/stats/arin/delegated-arin-extended-20220923. (2022).

[10] Todd Arnold, Matt Calder, Italo Cunha, Arpit Gupta, Harsha V. Madhyastha, Michael Schapira, and Ethan Katz-Bassett. 2019. Beating bgp is harder than we thought. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (HotNets '19). Association for Computing Machinery, Princeton, NJ, USA, 9–16. ISBN: 9781450370202. DOI: 10.1145/3365609.3365865. https://doi.org/10.1145/3365609.3365865.

[11] Electronic Arts. 2024. Need for speed: the run. https://www.ea.com/games/need-for-speed. (January 2024).

[12] David Barrera, Laurent Chuat, Adrian Perrig, Raphael M. Reischuk, and Pawel Szalachowski. 2017. The scion internet architecture. *Commun. ACM*, 60, 6, (May 2017), 56–65. ISSN: 0001-0782. DOI: 10.1145/3085591. https://doi.org/10.1145/3085591.

[13] Ahmed Bashandy, Clarence Filsfils, Stefano Previdi, Bruno Decraene, Stephane Litkowski, and Rob Shakir. 2019. Segment Routing with the MPLS Data Plane. RFC 8660. (December 2019). DOI: 10.17487/RFC8660. https://www.rfc-editor.org/info/rfc8660.

[14] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. 2004. The effects of loss and latency on user performance in unreal tournament 2003®. In *Proceedings of 3rd*

ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '04). Association for Computing Machinery, Portland, Oregon, USA, 144–151. ISBN: 158113942X. DOI: 10.1145/1016540.1016556. https://doi.org/10.1145/1016540.1016556.

[15] Steven Bellovin, Randy Bush, and David Ward. 2014. Security Requirements for BGP Path Validation. RFC 7353. (August 2014). DOI: 10.17487/RFC7353. https://www.rfc-editor.org/info/rfc7353.

[16] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. 2006. Colyseus: a distributed architecture for online multiplayer games. In *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3* (NSDI'06). USENIX Association, San Jose, CA, 12.

[17] 1991. Blizzard entertainment. website. https://www.blizzard.com/en-us/. (February 1991).

[18] Olivier Bonaventure, Quentin De Coninck, Fabien Duchêne, Anthony Gego, Mathieu Jadin, François Michel, Maxime Piraux, Chantal Poncin, and Olivier Tilmans. 2020. Open educational resources for computer networking. *ACM SIGCOMM Computer Communication Review*, 50, (August 2020), 38–45. DOI: 10.1145/3411740.3411746.

[19] 2022. Caida as rank. http://as-rank.caida.org/. (May 2022).

[20] 2023. CAIDA UCSD AS relationships dataset, 2020-02-01. https://www.caida.org/catalog/datasets/as-relationships/. (2023).

[21] 2022. CAIDA UCSD AS relationships dataset, 2022-09-01. https://www.caida.org/catalog/datasets/as-relationships/. (2022).

[22] 2023. CAIDA UCSD AS to organisation mapping dataset, 2023-04-01. https://www.caida.org/data/as-organizations/. (2023).

[23] 2022. CAIDA UCSD routeviews prefix to as mappings dataset for ipv4 and ipv6, 2022-09-22. https://www.caida.org/catalog/datasets/routeviews-prefix2as/. (2022).

[24] Juan Camilo Cardona, Stefano Vissicchio, Paolo Lucente, and Pierre Francois. 2016. "i can't get no satisfaction": helping autonomous systems identify their unsatisfied interdomain interests. *IEEE Transactions on Network and Service Management*, 13, 1, 43–57. DOI: 10.1109/TNSM.2016.2525003.

[25] Marc Carrascosa and Boris Bellalta. 2020. Cloud-gaming: analysis of google stadia traffic. *CoRR*, abs/2009.09786. arXiv: 2009.09786. https://arxiv.org/abs/2009.09786.

[26] Advanced Network Technology Center. [n. d.] Routeviews project. [Online]. Available: http://www.routeviews.org/routeviews/. [Accessed 16 April 2020]. ().

[27] Xianhui Che and Barry Ip. 2012. Packet-level traffic analysis of online games from the genre characteristics perspective. *Journal of Network and Computer Applications*, 35, 1, 240–252. Collaborative Computing and Applications. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2011.08.005. https://www.sciencedirect.com/science/article/pii/S1084804511001731.

[28] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. 2011. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM International Conference on Multimedia* (MM '11). Association for Computing Machinery, Scottsdale, Arizona, USA, 1269–1272. ISBN: 9781450306164. DOI: 10.1145/2072298.2071991. https://doi.org/10.1145/2072298.2071991.

[29] Meiling Chen and Li Su. 2023. The Requirements for Secure Routing Path. Internet-Draft draft-chen-secure-routing-requirements-02. Work in Progress. Internet Engineering Task Force, (November 2023). 4 pages. https://datatracker.ietf.org/doc/draft-chen-secure-routing-requirements/02/.

[30] Ze Chen, Jun Bi, Yonghong Fu, Yangyang Wang, and Anmin Xu. 2015. Mlv: a multi-dimension routing information exchange mechanism for inter-domain sdn. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, 438–445. DOI: 10.1109/ICNP.2015.34.

[31] Jaeyoung Choi, Jong Han Park, Pei-chun Cheng, Dorian Kim, and Lixia Zhang. 2011. Understanding bgp next-hop diversity. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 846–851. DOI: 10.1109/INFCOMW.2011.5928930.

[32] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. 2012. The brewing storm in cloud gaming: a measurement study on cloud to end-user latency. In *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, 1–6. DOI: 10.1109/NetGames.2012.6404024.

[33] Benoît Claise. 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954. (October 2004). DOI: 10.17487/RFC3954. https://www.rfc-editor.org/info/rfc3954.

[34] Mark Claypool. 2005. The effect of latency on user performance in real-time strategy games. *Computer Networks*, 49, 1, 52–70. Networking Issue in Entertainment Computing. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2005.04.008. https://www.sciencedirect.com/science/article/pii/S1389128605001003.

[35] Mark Claypool and Kajal Claypool. 2006. Latency and player actions in online games. en. *Communications of the ACM*, 49, 11, (November 2006), 40–45. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1167838.1167860. Retrieved 11/28/2023 from https://dl.acm.org/doi/10.1145/1167838.1167860.

[36] Mark Claypool and Kajal Claypool. 2010. Latency can kill: precision and deadline in online games. In (MMSys '10). Association for Computing Machinery, Phoenix, Arizona, USA, 215–222. ISBN: 9781605589145.

DOI: 10.1145/1730836.1730863. https://doi.org/10.1145/1730836.1730863.

[37] Steam Community. 2024. Sub-tick is just terrible. https://steamcommunity.com/app/730/discussions/0/3881597531954819040/. (January 2024).

[38] Valve Corporation. 2012. Counter strike: global offensive. game. https://store.steampowered.com/app/730/CounterStrike_Global_Offensive/. (2012).

[39] Matt deWet and David Straily. 2020. Peeking into valorant's netcode. riot games. https://technology.riotgames.com/news/peeking-valorants-netcode. (July 2020).

[40] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. 2021. A network analysis on cloud gaming: stadia, geforce now and psnow. *Network*, 1, 3, 247–260. ISSN: 2673-8732. DOI: 10.3390/network1030015. https://www.mdpi.com/2673-8732/1/3/15.

[41] Matthias Dick, Oliver Wellnitz, and Lars Wolf. 2005. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games* (NetGames '05). Association for Computing Machinery, Hawthorne, NY, 1–7. ISBN: 1595931562. DOI: 10.1145/1103599.1103624. https://doi.org/10.1145/1103599.1103624.

[42] Ragnhild Eg, Kjetil Raaen, and Mark Claypool. 2018. Playing with delay: with poor timing comes poor performance, and experience follows suit. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, 1–6. DOI: 10.1109/QoMEX.2018.8463382.

[43] Blizzard Entertainment. 2024. World of warcraft. https://worldofwarcraft.blizzard.com/en-us/. (January 2024).

[44] exitlag. 2009. Exitlag. https://www.exitlag.com/. (December 2009).

[45] Adrian Farrel, Olufemi Komolafe, and Seisho Yasukawa. 2009. An Analysis of Scaling Issues in MPLS-TE Core Networks. RFC 5439. (February 2009). DOI: 10.17487/RFC5439. https://www.rfc-editor.org/info/rfc5439.

[46] Dimeji Fayomi. 2023. Directed topology graph between atlas probes and game servers. https://github.com/olafayomi/OvW-Sys-Research/blob/main/Measurement-Results/AS_path_graph1.html. Accessed: 2024-11-04. (2023).

[47] Dimeji Fayomi. 2023. Overwatch and emulation implementation[source code]. https://github.com/olafayomi/OvW-Sys-Research. (2023).

[48] Dimeji Fayomi. 2023. Weighted directed topology graph between atlas probes and game servers with rtt measurements. https://github.com/olafayomi/OvW-Sys-Research/blob/main/Measurement-Results/AS_path_graph2.html. Accessed: 2024-11-04. (2023).

[49] Nick Feamster, Jared Winick, and Jennifer Rexford. 2004. A model of bgp routing for network engineering. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS '04/Performance '04). Association for Computing Machinery, New York, NY, USA, 331–342. ISBN: 1581138733. DOI: 10.1145/1005686.1005726. https://doi.org/10.1145/1005686.1005726.

[50] Mark Fedor, Martin Lee Schoffstall, James R. Davin, and Dr. Jeff D. Case. 1990. Simple Network Management Protocol (SNMP). RFC 1157. (May 1990). DOI: 10.17487/RFC1157. https://www.rfc-editor.org/info/rfc1157.

[51] Wu-chang Feng, F. Chang, Wu-chi Feng, and J. Walpole. 2005. A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, 13, 3, 488–500. DOI: 10.1109/TNET.2005.850221.

[52] Clarence Filsfils, Darren Dukes, Stefano Previdi, John Leddy, Satoru Matsushima, and Daniel Voyer. 2020. IPv6 Segment Routing Header (SRH). RFC 8754. (March 2020). DOI: 10.17487/RFC8754. https://www.rfc-editor.org/info/rfc8754.

[53] Clarence Filsfils, Stefano Previdi, Gaurav Dawra, Ebben Aries, and Dmitry Afanasiev. 2021. Segment Routing Centralized BGP Egress Peer Engineering. RFC 9087. (August 2021). DOI: 10.17487/RFC9087. https://www.rfc-editor.org/info/rfc9087.

[54] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. 2018. Segment Routing Architecture. RFC 8402. (July 2018). DOI: 10.17487/RFC8402. https://www.rfc-editor.org/info/rfc8402.

[55] Romain Fontugne, Cristel Pelsser, Emile Aben, and Randy Bush. 2017. Pinpointing delay and forwarding anomalies using large-scale traceroute measurements. In *Proceedings of the 2017 Internet Measurement Conference* (IMC '17). Association for Computing Machinery, London, United Kingdom, 15–28. ISBN: 9781450351188. DOI: 10.1145/3131365.3131384. https://doi.org/10.1145/3131365.3131384.

[56] Tobias Fritsch, Hartmut Ritter, and Jochen Schiller. 2005. The effect of latency and network limitations on mmorpgs: a field study of everquest2. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games* (NetGames '05). Association for Computing Machinery, Hawthorne, NY, 1–9. ISBN: 1595931562. DOI: 10.1145/1103599.1103623. https://doi.org/10.1145/1103599.1103623.

[57] Broadsword Online Games. 2024. Star wars: the old republic. https://www.swtor.com/. (January 2024).

[58] EA Games. 2024. Command & conquer: generals. https://www.ea.com/games/command-and-conquer. (January 2024).

[59] Riot Games. 2009. League of legends. game. https://www.leagueoflegends.com. (October 2009).

[60] Chen Gao, Haifeng Shen, and M. Ali Babar. 2016. Concealing jitter in multi-player online games through predictive behaviour modeling. In *2016 IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 62–67. DOI: 10.1109/CSCWD.2016.7565964.

[61] Lixin Gao and Jennifer Rexford. 2000. Stable internet routing without global coordination. In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS '00). Association for Computing Machinery, Santa Clara, California, USA, 307–317. ISBN: 1581131941. DOI: 10.1145/339331.339426. https://doi.org/10.1145/339331.339426.

[62] Inmaculada García and Ramón Mollá. 2005. Improving quality of service in videogames. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology* (ACE '05). Association for Computing Machinery, Valencia, Spain, 250–253. ISBN: 1595931104. DOI: 10.1145/1178477.1178519. https://doi.org/10.1145/1178477.1178519.

[63] Steven Gargolinski, Christopher St. Pierre, and Mark Claypool. 2005. Game server selection for multiple players. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games* (NetGames '05). Association for Computing Machinery, Hawthorne, NY, 1–6. ISBN: 1595931562. DOI: 10.1145/1103599.1103616. https://doi.org/10.1145/1103599.1103616.

[64] P. Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. 2009. Pathlet routing. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (SIGCOMM '09). Association for Computing Machinery, Barcelona, Spain, 111–122. ISBN: 9781605585949.

DOI: 10.1145/1592568.1592583. https://doi.org/10.1145/1592568.1592583.

[65] Philippe Graff, Xavier Marchal, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu, and Olivier Festor. 2021. An analysis of cloud gaming platforms behavior under different network constraints. In *2021 17th International Conference on Network and Service Management (CNSM)*, 551–557. DOI: 10.23919/CNSM52442.2021.9615562.

[66] Mihovil Grguric. 2024. Mobile game session length: how to track & increase it. https://www.blog.udonis.co/mobile-marketing/mobile-games/session-length. (June 2024).

[67] 2016. Grpc. https://grpc.io/. (2016).

[68] 2016. Grpc. https://protobuf.dev/. (2016).

[69] Gufo Labs and Project's Contributors. [n. d.] Gufo Ping. ().

[70] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*. Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors. Pasadena, CA USA, 11–15.

[71] David Halbhuber, Valentin Schwind, and Niels Henze. 2022. Don't break my flow: effects of switching latency in shooting video games. *Proc. ACM Hum.-Comput. Interact.*, 6, CHI PLAY, Article 229, (October 2022), 20 pages. DOI: 10.1145/3549492. https://doi.org/10.1145/3549492.

[72] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. 2012. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (CoNEXT '12). Association for Computing Machinery, Nice, France, 253–264. ISBN: 9781450317757. DOI: 10.1145/2413176.2413206. https://doi.org/10.1145/2413176.2413206.

[73] Nicholas Hart, Charalampos Rotsos, Vasileios Giotsas, Nicholas Race, and David Hutchison. 2020. Λbgp: rethinking bgp programmability. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 1–9. DOI: 10.1109/NOMS47738.2020.9110331.

[74] Jiayue He and Jennifer Rexford. 2008. Toward internet-wide multipath routing. *IEEE Network*, 22, 2, 16–21. DOI: 10.1109/MNET.2008.4476066.

[75] Stephanie Heintz and Effie Lai-Chong Law. 2015. The game genre map: a revised game classification. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play* (CHI PLAY '15). Association for Computing Machinery, , London, United Kingdom, 175–184. ISBN: 9781450334662. DOI: 10.1145/2793107.2793123. https://doi.org/10.1145/2793107.2793123.

[76] Carl Hewitt. 1977. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8, 3, 323–364. ISSN: 0004-3702. DOI: {https://doi.org/10.1016/0004-3702(77)90033-9}.

[77] Oliver Hohlfeld, Hannes Fiedler, Enric Pujol, and Dennis Guse. 2016. Insensitivity to network delay: minecraft gaming experience of casual gamers. In *2016 28th International Teletraffic Congress (ITC 28)*. Volume 03, 31–33. DOI: 10.1109/ITC-28.2016.313.

[78] Eben Howard, Clint Cooper, Mike P. Wittie, Steven Swinford, and Qing Yang. 2014. Cascading impact of lag on quality of experience in cooperative multiplayer games. In *2014 13th Annual Workshop on Network and Systems Support for Games*, 1–6. DOI: 10.1109/NetGames.2014.7008965.

[79] Geoff Huston. 2001. Analysing the internet's bgp routing table. *The Internet Protocol Journal*, 4, 1, 2–15.

[80] Ward Infinity, Treyarch, Sledgehammer Games, and Raven Software. 2003. Call of duty. game. (2003).

[81] December. Ip location. https://www.iplocation.net/. (2023 December).

[82] Jared Jardine and Daniel Zappala. 2008. A hybrid architecture for massively multiplayer online games. In *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games* (NetGames '08). Association for Computing Machinery, Worcester, Massachusetts, 60–65. ISBN: 9781605581323. DOI: 10.1145/1517494.1517507. https://doi.org/10.1145/1517494.1517507.

[83] Kristine Jørgensen. 2013. *Gameworld Interfaces*. The MIT Press. ISBN: 0262026864.

[84] Koichiro Kanaya, Yasunobu Toyota, Wataru Mishima, Hiroki Shirokura, and Hiroshi Esaki. 2022. Qoe-aware content oriented path optimisation framework with egress peer engineering. In *2022 Tenth International Symposium on Computing and Networking (CANDAR)*, 36–45. DOI: 10.1109/CANDAR57322.2022.00013.

[85] Abdul Malik Khan, Ivica Arsov, Marius Preda, Sophie Chabridon, and Antoine Beugnard. 2010. Adaptable client-server architecture for mobile multiplayer games. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques* (SIMUTools '10) Article 11. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Torremolinos, Malaga, Spain, 7 pages. ISBN: 9789639799875. DOI: 10.4108/ICST.SIMUTOOLS2010.8704. https://doi.org/10.4108/ICST.SIMUTOOLS2010.8704.

[86] Joohwan Kim, Pyarelal Knowles, Josef Spjut, Ben Boudaoud, and Morgan Mcguire. 2020. Post-render warp with late input sampling improves aiming under high latency conditions. *Proc. ACM Comput. Graph. Interact. Tech.*, 3, 2, Article 12, (August 2020), 18 pages. DOI: 10.1145/3406187. https://doi.org/10.1145/3406187.

[87] Nate Kushman, Srikanth Kandula, and Bruce M. Maggs. 2007. R-BGP: staying connected in a connected world. In *4th USENIX Symposium*

on Networked Systems Design & Implementation (NSDI 07). USENIX Association, Cambridge, MA, (April 2007). https://www.usenix.org/conference/nsdi-07/r-bgp-staying-connected-connected-world.

[88] 2022. LACNIC extended delagation statistics, 2022-09-22. https://ftp.lacnic.net/pub/stats/apnic/2022/delegated-apnic-extended-20220922.gz. (2022).

[89] Asif Ali Laghari, Hui He, Kamran Ali Memon, Rashid Ali Laghari, Imtiaz Ali Halepoto, and Asiya Khan. 2019. Quality of experience (qoe) in cloud gaming models: a review. *multiagent and grid systems*, 15, 3, 289–304.

[90] Nicolas LaLone, Phoebe O. Toups Dugas, and Michelle V. Cormier. 2023. A quest?!: the secret life of gameworld punctuation. *Proc. ACM Hum.-Comput. Interact.*, 7, CHI PLAY, Article 394, (October 2023), 32 pages. DOI: 10.1145/3611040. https://doi.org/10.1145/3611040.

[91] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (Hotnets-IX) Article 19. Association for Computing Machinery, Monterey, California, 6 pages. ISBN: 9781450304092. DOI: 10.1145/1868447.1868466. https://doi.org/10.1145/1868447.1868466.

[92] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2018. Internet anycast: performance, problems, and potential. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (SIGCOMM '18). Association for Computing Machinery, Budapest, Hungary, 59–73. ISBN: 9781450355674. DOI: 10.1145/3230543.3230547. https://doi.org/10.1145/3230543.3230547.

[93] Shengmei Liu, Mark Claypool, Atsuo Kuwahara, James Scovell, and Jamie Sherman. 2021. L33t or n00b? how player skill alters the effects

of network latency on first person shooter game players. In *Proceedings of the Workshop on Game Systems (GameSys '21)* (GameSys '21). Association for Computing Machinery, Istanbul, Turkey, 1–6. ISBN: 9781450384377. DOI: 10.1145/3458335.3460811. https://doi.org/10.1145/3458335.3460811.

[94] Shengmei Liu, Mark Claypool, Atsuo Kuwahara, James Scovell, and Jamie Sherman. 2021. The effects of network latency on competitive first-person shooter game players. In *2021 13th International Conference on Quality of Multimedia Experience (QoMEX)*, 151–156. DOI: 10.1109/QoMEX51781.2021.9465419.

[95] Shengmei Liu, Xiaokun Xu, and Mark Claypool. 2022. A survey and taxonomy of latency compensation techniques for network computer games. *ACM Comput. Surv.*, 54, 11s, Article 243, (September 2022), 34 pages. ISSN: 0360-0300. DOI: 10.1145/3519023. https://doi.org/10.1145/3519023.

[96] Sharat Chandra Madanapalli, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2022. Know thy lag: in-network game detection and latency measurement. In *Passive and Active Measurement: 23rd International Conference, PAM 2022, Virtual Event, March 28–30, 2022, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 395–410. ISBN: 978-3-030-98784-8. DOI: 10.1007/978-3-030-98785-5_17. https://doi.org/10.1007/978-3-030-98785-5_17.

[97] Thomas Mangin. 2013. ExaBGP. https://github.com/Exa-Networks/exabgp.git. (2013).

[98] Marc Manzano, Manuel Uruena, Mirko Sužnjević, Eusebi Calle, Jose Alberto Hernandez, and Maja Matijasevic. 2014. Dissecting the protocol and network traffic of the onlive cloud gaming platform. *Multimedia systems*, 20, 5, 451–470.

[99] 2023. Maxmind geoip. https://www.maxmind.com/en/geoip-demo. (2023).

[100] Peyton Maynard-Koran. 2015. Fixing the internet for real time applications: part i. https://technology.riotgames.com/news/fixing-internet-real-time-applications-part-i. (November 2015).

[101] Peyton Maynard-Koran. 2016. Riot direct: video. https://technology.riotgames.com/news/riot-direct-video. (May 2016).

[102] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38, 2, (March 2008), 69–74. ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. https://doi.org/10.1145/1355734.1355746.

[103] Florian Metzger, Stefan Geißler, Alexej Grigorjew, Frank Loh, Christian Moldovan, Michael Seufert, and Tobias Hoßfeld. 2022. An introduction to online video game qos and qoe influencing factors. *IEEE Communications Surveys & Tutorials*, 24, 3, 1894–1925. DOI: 10.1109/COMST.2022.3177251.

[104] Miniplay. 2024. Minesweeper. https://www.miniplay.com/game/minesweeper-io. (January 2024).

[105] Ryo Nakamura, Kazuki Shimizu, Teppei Kamata, and Cristel Pelsser. 2022. A first measurement with bgp egress peer engineering. In *Passive and Active Measurement*. Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser, editors. Springer International Publishing, Cham, 199–215. ISBN: 978-3-030-98785-5.

[106] 2006. *Network latency, jitter and loss. Networking and Online Games*. John Wiley & Sons, Ltd. Chapter 5, 69–82. ISBN: 9780470030479. DOI: https://doi.org/10.1002/047003047X.ch5. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047003047X.ch5.

https://onlinelibrary.wiley.com/doi/abs/10.1002/047003047X.ch5.

[107] James Nichols and Mark Claypool. 2004. The effects of latency on online madden nfl football. In *Proceedings of the 14th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV '04). Association for Computing Machinery, Cork, Ireland, 146–151. ISBN: 1581138016. DOI: 10.1145/1005847.1005879. https://doi.org/10.1145/1005847.1005879.

[108] 2023. Nvidia cumulus linux. website. https://www.nvidia.com/en-us/networking/ethernet-switching/cumulus-linux/. (2023).

[109] Chiara Orsini, Alistair King, Danilo Giordano, Vasileios Giotsas, and Alberto Dainotti. 2016. Bgpstream: a software framework for live and historical bgp data analysis. In *Proceedings of the 2016 Internet Measurement Conference* (IMC '16). Association for Computing Machinery, Santa Monica, California, USA, 429–444. ISBN: 9781450345262. DOI: 10.1145/2987443.2987482. https://doi.org/10.1145/2987443.2987482.

[110] Lothar Pantel and Lars C. Wolf. 2002. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV '02). Association for Computing Machinery, Miami, Florida, USA, 23–29. ISBN: 1581135122. DOI: 10.1145/507670.507674. https://doi.org/10.1145/507670.507674.

[111] PeeringDB. 2022. PeeringDB. IXPs and colocation database, 2022-09-22. https://www.peeringdb.com. (2022).

[112] P. Psenak, C. Filsfils, A. Bashandy, B. Decraene, and Z. Hu. 2023. Rfc 9352: is-is extensions to support segment routing over the ipv6 data plane. USA, (2023).

[113] Himabindu Pucha, Ying Zhang, Z. Morley Mao, and Y. Charlie Hu. 2007. Understanding network delay changes caused by routing events. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (SIGMETRICS '07). Association for Computing Machinery, San Diego, California, USA, 73–84. ISBN: 9781595936394. DOI: 10.1145/1254882.1254891. https://doi.org/10.1145/1254882.1254891.

[114] Donghong Qin, Jaihai Yang, Zhuolin Liu, Hui Wang, Bin Zhang, and Wei Zhang. 2012. Amir: another multipath interdomain routing. In *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, 581–588. DOI: 10.1109/AINA.2012.83.

[115] Prathy Raman and Marcel Flores. 2021. Building out the basics with hoplets. In *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29 - April 1, 2021, Proceedings*. Springer-Verlag, Cottbus, Germany, 355–370. ISBN: 978-3-030-72581-5. DOI: 10.1007/978-3-030-72582-2_21. https://doi.org/10.1007/978-3-030-72582-2_21.

[116] 2006. *Recent online and multiplayer games. Networking and Online Games*. John Wiley & Sons, Ltd. Chapter 5, 69–82. ISBN: 9780470030479. DOI: https://doi.org/10.1002/047003047X.ch5. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047003047X.ch5. https://onlinelibrary.wiley.com/doi/abs/10.1002/047003047X.ch5.

[117] Yakov Rekhter, Susan Hares, and Tony Li. 2006. A Border Gateway Protocol 4 (BGP-4). RFC 4271. (January 2006).

[118] Jennifer Rexford and Constantine Dovrolis. 2010. Future internet architecture: clean-slate versus evolutionary research. *Commun. ACM*, 53, 9, (September 2010), 36–40. ISSN: 0001-0782. DOI: 10.1145/1810891.1810906. https://doi.org/10.1145/1810891.1810906.

[119] 2022. RIPE extended delagation statistics, 2022-09-23. https://ftp.ripe.net/ripe/stats/2022/delegated-ripencc-extended-20220923.bz2. (2022).

[120] 2023. Routing information service. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris. (September 2023).

[121] Katie Salen and Eric Zimmerman. 2003. *Rules of Play: Game Design Fundamentals*. The MIT Press. ISBN: 0262240459.

[122] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. 1999. The end-to-end effects of internet path selection. *SIGCOMM Comput. Commun. Rev.*, 29, 4, (August 1999), 289–299. ISSN: 0146-4833. DOI: 10.1145/316194.316233. https://doi.org/10.1145/316194.316233.

[123] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering egress with edge fabric: steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (SIGCOMM '17). Association for Computing Machinery, Los Angeles, CA, USA, 418–431. ISBN: 9781450346535. DOI: 10.1145/3098822.3098853. https://doi.org/10.1145/3098822.3098853.

[124] Andreas Schmid, David Halbhuber, Thomas Fischer, Raphael Wimmer, and Niels Henze. 2023. Small latency variations do not affect player performance in first-person shooters. *Proc. ACM Hum.-Comput. Interact.*, 7, CHI PLAY, Article 381, (October 2023), 20 pages. DOI: 10.1145/3611027. https://doi.org/10.1145/3611027.

[125] John Scudder, Rex Fernando, and Stephen Stuart. 2016. BGP Monitoring Protocol (BMP). RFC 7854. (June 2016). DOI: 10.17487/RFC7854. https://www.rfc-editor.org/info/rfc7854.

[126] 2023. Service router linux nos. website. https://www.nokia.com/networks/ip-networks/service-router-linux-NOS/. (2023).

[127] Ryan Shea, Jiangchuan Liu, Edith C.-H. Ngai, and Yong Cui. 2013. Cloud gaming: architecture and performance. *IEEE Network*, 27, 4, 16–21. DOI: 10.1109/MNET.2013.6574660.

[128] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. 2003. The effect of latency on user performance in warcraft iii. In *Proceedings of the 2nd Workshop on Network and System Support for Games* (NetGames '03). Association for Computing Machinery, Redwood City, California, 3–14. ISBN: 1581137346. DOI: 10.1145/963900.963901. https://doi.org/10.1145/963900.963901.

[129] Walber José Adriano Silva and Djamel Fawzi Hadj Sadok. 2018. A survey on efforts to evolve the control plane of inter-domain routing. *Information*, 9, 5. ISSN: 2078-2489. DOI: 10.3390/info9050125. https://www.mdpi.com/2078-2489/9/5/125.

[130] Sandeep Kumar Singh, Tamal Das, and Admela Jukan. 2015. A survey on internet multipath routing and provisioning. *IEEE Communications Surveys & Tutorials*, 17, 4, 2157–2175. DOI: 10.1109/COMST.2015.2460222.

[131] Carolina Road Software. 2024. Jigsaw puzzle explorer. https://www.jigsawexplorer.com/. (January 2024).

[132] 2023. Source multiplayer networking. https://developer.valvesoftware.com/w/index.php?title=Source_Multiplayer_Networking&oldid=328825. (July 2023).

[133] EA Sports. 2024. Fifa 23. https://www.ea.com/games/fifa/fifa-23. (January 2024).

[134] Neil Spring, Ratul Mahajan, and Thomas Anderson. 2003. The causes of path inflation. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*

(SIGCOMM '03). Association for Computing Machinery, Karlsruhe, Germany, 113–124. ISBN: 1581137354. DOI: 10.1145/863955.863970. https://doi.org/10.1145/863955.863970.

[135] RIPE Ncc Staff. 2015. Ripe atlas: a global internet measurement network. *Internet Protocol Journal*, 18, 3, 2–26.

[136] Statista. 2023. Statista market insights: online games - worldwide. https://www.statista.com/outlook/dmo/digital-media/video-games/online-games/worldwide. (July 2023).

[137] Microsft Game Studios. 2024. Age of mythology. https://www.ageofempires.com/games/aom/. (January 2024).

[138] Mojang Studios. 2024. Minecraft. https://www.minecraft.net/en-us. (January 2024).

[139] Subspace. 2023. Subspace. https://subspace.com/. (August 2023).

[140] Renata Teixeira, Keith Marzullo, Stefan Savage, and Geoffrey M. Voelker. 2003. In search of path diversity in isp networks. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement* (IMC '03). Association for Computing Machinery, Miami Beach, FL, USA, 313–318. ISBN: 1581137737. DOI: 10.1145/948205.948247. https://doi.org/10.1145/948205.948247.

[141] 1986. Ubisoft entertainment. website. https://www.ubisoft.com/en-au/. (March 1986).

[142] 1986. Valve corporation. website. https://www.valvesoftware.com/en/. (March 1986).

[143] Pier Luigi Ventre, Stefano Salsano, Marco Polverini, Antonio Cianfrani, Ahmed Abdelsalam, Clarence Filsfils, Pablo Camarillo, and Francois Clad. 2021. Segment routing: a comprehensive survey of research activities, standardization efforts, and implementation results. *IEEE Communications Surveys & Tutorials*, 23, 1, 182–221. DOI: 10.1109/COMST.2020.3036826.

[144] Pier Luigi Ventre, Mohammad Mahdi Tajiki, Stefano Salsano, and Clarence Filsfils. 2018. SDN architecture and southbound apis for ipv6 segment routing enabled wide area networks. *CoRR*, abs/1810.06008. arXiv: 1810.06008. http://arxiv.org/abs/1810.06008.

[145] Arun Viswanathan, Eric C. Rosen, and Ross Callon. 2001. Multiprotocol Label Switching Architecture. RFC 3031. (January 2001). DOI: 10.17487/RFC3031. https://www.rfc-editor.org/info/rfc3031.

[146] Daniel Walton, Alvaro Retana, Enke Chen, and John Scudder. 2016. Advertisement of Multiple Paths in BGP. RFC 7911. (July 2016). DOI: 10.17487/RFC7911. https://www.rfc-editor.org/info/rfc7911.

[147] Calvin Wan, Ronnie Cheung, Simon Wong, and Mei Po Ng. 2012. A communicaton system for massive multiplayer online game. In *2012 IEEE International Conference on Signal Processing, Communication and Computing (ICSPCC 2012)*, 400–405. DOI: 10.1109/ICSPCC.2012.6335623.

[148] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. 2006. A measurement study on the impact of routing events on end-to-end internet path performance. *SIGCOMM Comput. Commun. Rev.*, 36, 4, (August 2006), 375–386. ISSN: 0146-4833. DOI: 10.1145/1151659.1159956. https://doi.org/10.1145/1151659.1159956.

[149] Lan Wang, Xiaoliang Zhao, Dan Pei, Randy Bush, Daniel Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. 2002. Observation and analysis of bgp behavior under stress. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment* (IMW '02). Association for Computing Machinery, Marseille, France, 183–195. ISBN: 158113603X. DOI: 10.1145/637201.637231. https://doi.org/10.1145/637201.637231.

[150] Yangyang Wang, Jun Bi, Pingping Lin, Yikai Lin, and Keyao Zhang. 2016. Sdi: a multi-domain sdn mechanism for fine-grained inter-domain

routing. *Annals of Telecommunications*, 71, (April 2016). DOI: 10.1007/s12243-016-0513-z.

[151] Yangyang Wang, Jun Bi, Keyao Zhang, and Yangchun Wu. 2016. A framework for fine-grained inter-domain routing diversity via sdn. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 751–756. DOI: 10.1109/ICUFN.2016.7537139.

[152] WTFast. 2009. Wtfast. https://www.wtfast.com/. (December 2009).

[153] Li Xiao, King-Shan Lui, Jun Wang, and K. Nahrstedt. 2002. Qos extension to bgp. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.* 100–109. DOI: 10.1109/ICNP.2002.1181390.

[154] Xiaokun Xu and Mark Claypool. 2022. Measurement of cloud-based game streaming system response to competing tcp cubic or tcp bbr flows. In *Proceedings of the 22nd ACM Internet Measurement Conference* (IMC '22). Association for Computing Machinery, Nice, France, 305–316. ISBN: 9781450392594. DOI: 10.1145/3517745.3561464. https://doi.org/10.1145/3517745.3561464.

[155] Xiaokun Xu and Mark Claypool. 2022. Measurement of the responses of cloud-based game streaming to network congestion. In *Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video* (NOSSDAV '22). Association for Computing Machinery, , Athlone, Ireland, 22–28. ISBN: 9781450393836. DOI: 10.1145/3534088.3534346. https://doi.org/10.1145/3534088.3534346.

[156] Xiaokun Xu, Shengmei Liu, and Mark Claypool. 2022. The effects of network latency on counter-strike: global offensive players. In *2022 14th International Conference on Quality of Multimedia Experience (QoMEX)*, 1–6. DOI: 10.1109/QoMEX55416.2022.9900915.

[157] Amir Yahyavi and Bettina Kemme. 2013. Peer-to-peer architectures for massively multiplayer online games: a survey. *ACM Comput. Surv.*, 46, 1, Article 9, (July 2013), 51 pages. ISSN: 0360-0300. DOI: 10.1145/2522968.2522977. https://doi.org/10.1145/2522968.2522977.

[158] Xiaowei Yang, David Clark, and Arthur W. Berger. 2007. Nira: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 15, 4, 775–788. DOI: 10.1109/TNET.2007.893888.

[159] M. Yannuzzi, X. Masip-Bruin, and O. Bonaventure. 2005. Open issues in interdomain routing: a survey. *IEEE Network*, 19, 6, 49–56. DOI: 10.1109/MNET.2005.1541721.

[160] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2017. Taking the edge off with espresso: scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (SIGCOMM '17). Association for Computing Machinery, Los Angeles, CA, USA, 432–445. ISBN: 9781450346535. DOI: 10.1145/3098822.3098854. https://doi.org/10.1145/3098822.3098854.

[161] Marcelo Zamith, Mark Joselli, Luis Valente, Esteban Clua, Anselmo Montenegro, Regina Celia P Leal-Toledo, and Bruno Feijo. 2009. A game loop architecture with automatic distribution of tasks and load balancing between processors. *Proceedings of SBGames*, 5–8.

[162] Sebastian Zander, Ian Leeder, and Grenville Armitage. 2005. Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology* (ACE '05). Association

for Computing Machinery, Valencia, Spain, 117–124. ISBN: 1595931104. DOI: 10.1145/1178477.1178493. https://doi.org/10.1145/1178477.1178493.

[163] Dapeng Zhu, Mark Gritter, and David R. Cheriton. 2003. Feedback based routing. *SIGCOMM Comput. Commun. Rev.*, 33, 1, (January 2003), 71–76. ISSN: 0146-4833. DOI: 10.1145/774763.774774. https://doi.org/10.1145/774763.774774.

Appendix A

Latency Measurement Message Schema

```
syntax = "proto3";

package perfmon;

message ExitNode {
  string name = 1;
  string address = 2;
  float delay = 3;
  float estDelay = 4;
  float loss = 5;
  float devDelay = 6;
}

// Performance message to PARMetricsModule containing
// measurements via the available ExitNodes to a
// destination
message PerformanceMsg {
  repeated ExitNode node = 1;
```

```
}

// Message to PARMetricsModule containing
// measurements via available ExitNodes
message DstMsmMsg {
  string DstAddr = 1;
  repeated ExitNode node = 2;
}

// Send multiple destinations in one message
message DstMsmMsgs {
  repeated DstMsmMsg dstMsm = 1;
}

// Client destinations to perform measurements
message Destinations {
  repeated string address = 1;
}

// RTT Measurements for each destination
message DstRTT {
  string address = 1;
  repeated float rtt = 2;
}

// RTT measurements messages for a set destinations
message DstRTTMsgs {
  repeated DstRTT dstrtt = 1;
}
```

```
// RTT Measurements for each client
message ClientRTT {
  string address = 1;
  float ertt = 2;
  float rtt = 3;
  float drtt = 4;
}

// RTT measurements messages for a set destinations
message ClRTTMsgs {
  repeated ClientRTT cl_rtt = 1;
}
```