D.1.1: Low-latency: applications, network solutions, attacks and optimisation techniques

Bertrand Mathieu[1], Stéphane Tuffin[1], Guillaume Doyen[2], Marius Letourneau[2], Hichem Magnouche[2], Thibault Cholez[3], Philippe Graff[3], Edgardo Montes de Oca[4], and Huu Nghia Nguyen[4]

[1]Orange
[2]UTT
[3]Loria
[4]Montimage

Abstract

The MOSAICO project aims at defining networking solutions for the secure and efficient delivery of low-latency applications, such as cloud gaming, cloud robotics, drone piloting, etc.. This first deliverable presents a survey and our analysis of all the key topics addressed by the project as well as a first selection of the candidate technologies. First, the different classes of low-latency applications are described, mainly focusing on their main characteristics and their latency considerations. The sources of latency for delivering a service from the server to the end-clients are presented, highlighting the network ones, which we will deeply investigate and improve subsequently. To that aim, the deliverable introduces the main current network solutions for addressing the latency issue: queue management and control congestion algorithms, and Network Virtualisation Function (NFV) orchestration for deploying network both monolithic and micro services. To ensure the right delivery of low-latency applications, the main threats which could have an impact on latency are presented. Finally, the two main technologies we advocate in the project for implementing and deploying microservices ensuring the secure and efficient delivery of low-latency services are presented: OpenNetVM, a microservice technology, and P4 (Programming Protocol-Independent Packet Processor), a network data plane programming solution.

Contents

1 Introduction

The broad family of low-latency (LL) applications, such as cloud gaming, live video streaming, drone piloting, cloud robotics and tele-robotics are very demanding in terms of network latency, down to a few milliseconds. Even if they all can be classified under the umbrella of LL applications, they differ in various features: required network delay, necessary throughput, encoding schemes, server processing for negative latency, adapted transport protocol, management of variable network conditions, etc. It is then necessary to survey these LL applications in order to clearly identify their main characteristics. Furthermore, willing to define solutions not for only one application, is a crucial task to identify if some applications share common behaviours and can then be regrouped inside a same class. Investigating solutions for one (or several) classes(s) will then allow us to apply our MOSAICO solutions to a wider range of applications.

In the project, one key issue is the ontime delivery of low-latency applications. However, we should first identify where, when and how latency can increase or decrease in the end-to-end service delivery. This helps focusing on the more impacting parts when trying to improve the processing and the forwarding of the packets so as to ensure the required low latency. Being a networking project, we will mainly focus our activity in the networking related causes of latency, namely the structural delay and the interaction between endpoints and along the transmission paths. To that aim, we envision an orchestrator able to dynamically handle the placement of microservices, into a P4-based programmable network equipment or into OpenNetVM-based Virtual machines. This will help to reduce the latency induced by the network itself. Secondly, we aim to propose solutions with Active Queue Management (AQM) and Explicit Congestion Notification (ECN) mechanisms to help the applications to better manage variable network conditions and reduce their latency.

Another key issue of the project is the security of the LL applications. This topic is not yet largely investigated in the research community and we should then identify why and how security for LL applications differ from other applications, which threats/attacks can happen in the network preventing the good delivery of LL applications and how to detect as well as eventually mitigate them.

This deliverable presents the related work on these topics and the first selection of use-cases, network solutions and technology candidates we will work on. First, a detailed classification of the low-latency applications is described in section 2. It especially focuses on their main characteristics and their latency considerations. Then, we introduce the main sources of latency for delivering a service from the server to the end-clients, highlighting the causes that will be addressed in the project, i.e. the network related causes. Based-on the identified sources of latency and the characteristics of the LL applications, we classify them into three main categories that the MOSAICO project will further investigate. In section 4, we introduce the main current networking solutions addressing the latency issue: firstly those related to queue management and control congestion algorithms, and then those related to Network Virtualisation Function (NFV) orchestration. In section 5, we present the main threats which could have an impact on latency for delivering LL services. Finally, in section 6, the two main network solutions we advocate in the project for implementing and deploying microservices ensuring secure and efficient delivery of LL services are presented: OpenNetVM, a microservice technology, and P4 (Programming Protocol-Independent Packet Processor), a network data plane programming solution. These two technologies will be the pillars of our solutions in MOSAICO for offering high performing and secure LL services.

2 Classification of low-latency applications

In this section, we mention the related work of the applications pertaining for the MOSAICO project, following the classification, depicted in Figure 1. In the case of applications involving humans (i.e. interactive applications between humans, and interaction of humans with systems), network performance requirements are typically derived empirically from quality of experience measurements, taking into account the performances of other subsystems (e.g. media encoding/decoding). As human perceptions can vary between individuals or can depend on their recent history or on the surrounding environment, network performance objectives, and specifically latency, is often specified as a range of acceptable latencies. When the lower bound is reached there is no more notable improvements in the quality of experience while, when the upper bound is reached, an important part of the population will consider the quality of experience as unacceptable. An interesting point regarding the latency requirements of interactive applications involving humans is that network latency requirements will be in the same order of magnitude as human's physiological "latency" (muscular, audio, visual and haptics, i.e. tactile and kinesthetic): from milliseconds to hundreds of milliseconds magnitudes (for kinesthetic feedback).

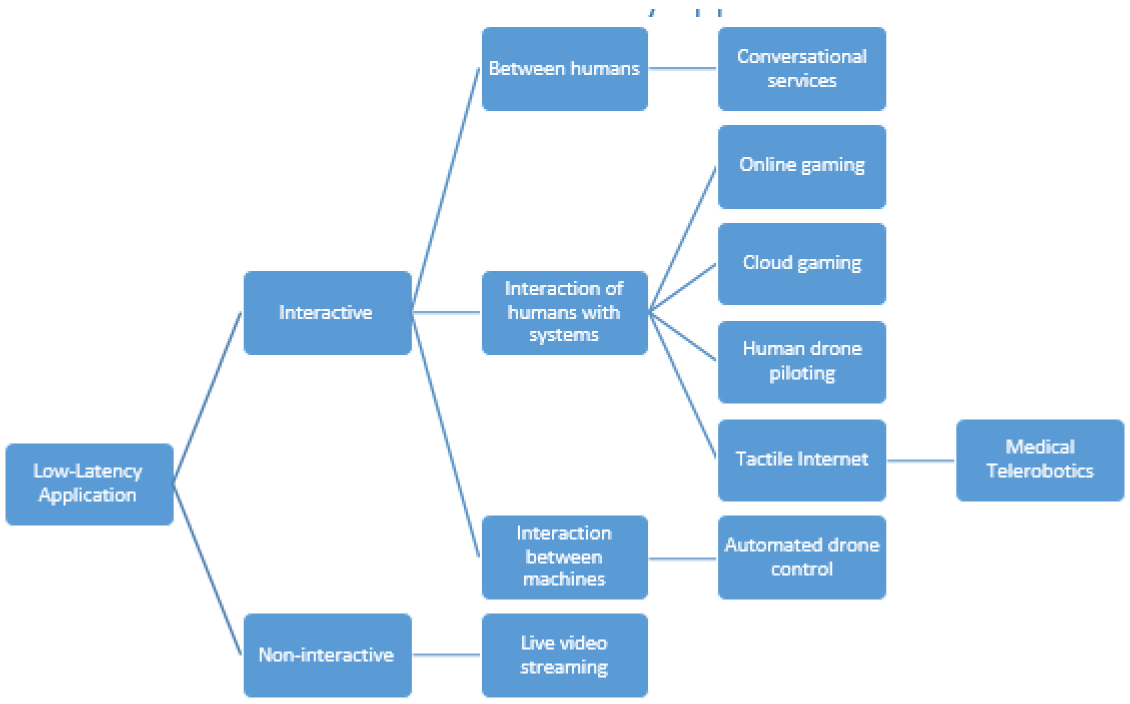


Figure 1: Classification of considered low-latency services

In the case of interactive applications between machines, the required network latency can usually be assessed more precisely. When the network latency bound is respected, the system composed of interacting machines works as expected, while, when it is not respected, system failures happen. The case of low-latency non-interactive applications refers to cases where the usefulness of the content that is transmitted through the network gets outdated as time passes. A typical example being the case of the live streaming [1] of a sport event such as a football match where viewers prefer to receive content (e.g. when a goal is being scored) no later than the others and low-delay video streaming is meant to be competitive with other broadcasting means (e.g. DVB-T) regarding the delay-to-live [2].

In this document, an application is considered as a low-latency application when the latency expected from networks is deemed to be challenging as compared to the current state of the art in deployed networks. This does not mean that latency needs are never respected but rather that there are some significant cases where the latency needs are not respected. There is a range of reasons why latency bounds can be respected by the network in some configurations but are not ensured in another. For the Internet, a landmark survey of the sources of network latencies can be found in [36]. However, it should be noticed that this excellent survey scarcely addresses the specificity of wireless networks such as cellular and Wi-Fi networks [3].

2.1 Online gaming

In online games, players interact through the Internet. The communication architectures supporting those interactions are manifold, as shown in Figure 2. In these architectures, all communications experience network latency. As latency is inevitable and usually not controllable by the players or the game editors, several latency compensation techniques have been designed to cope with it [137]. Prediction can be used to provide an impression of high responsiveness at the cost of abrupt changes when there is a discrepancy between the client and server states that needs to be fixed. Another latency compensation technique involves time manipulation. This can take the form of the introduction of an additional delay at the application level in order to equalize delay between players or deal with network jitter. Another case of time manipulation is time warp which consist in rolling back events at the server to incorporate the effect

[1] See also an expanded explanation here: https://aws.amazon.com/fr/media/tech/video-latency-in-live-streaming/

[2] Traditional video broadcast is often in the order of 6 seconds, partly due to the introduction of an artificial delay used for censorship as explained in: https://mux.com/blog/the-low-latency-live-streaming-landscape-in-2019/

[3] Wireless medium acquisition delays and wireless link error recovery delays are surveyed at a high level while the scheduling of wireless resources is ignored

of an action that arrived with some delay.

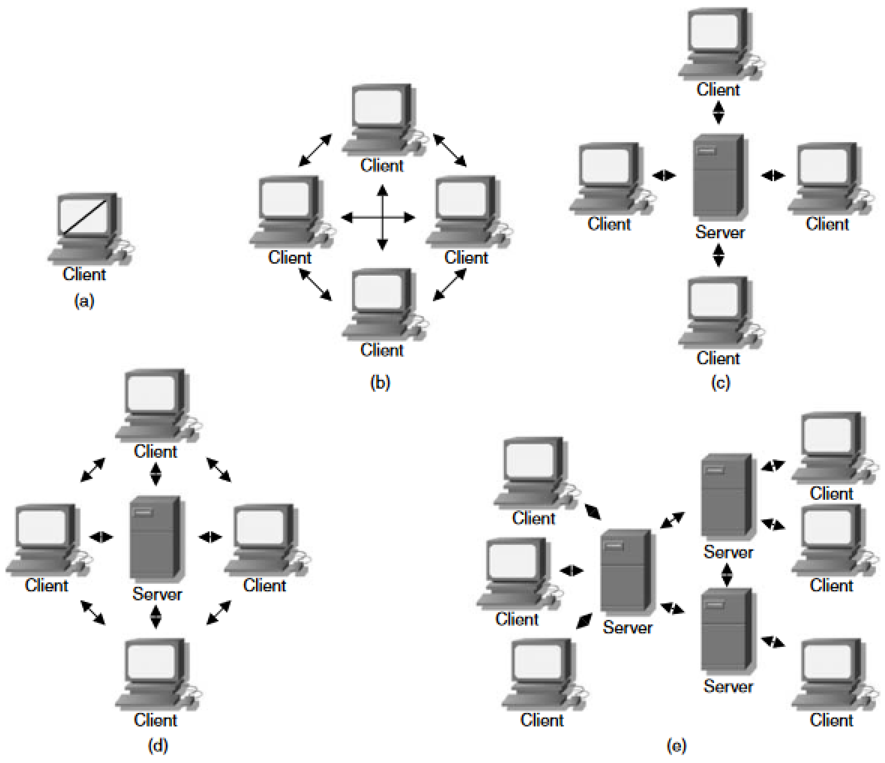


Figure 2: The possible communication architectures for multiplayer and network games : a) single screen, possibly 'split', on a single computer, b) peer-to-peer, c) a client-server, d) peer-to-peer-client server hybrid, and e) network of servers" [137]

The performance (e.g. number of frags in a FPS game) of online players has been shown to depend on network latency. In [137] several such studies are cited by the authors. For instance, with Quake III Arena, a study is cited showing frag rates of 2.5 − 3 per minute for players with 50ms of median ping latency to decrease almost linearly to 1 per minute for players with 400ms of media ping latency. For Half-Life, another study is cited concluding that a latency of around 225ms − 250ms dissuades players from joining a server while below this level the impact of latencies on player departure behavior was not noticeable. Regarding Unreal Tournament 2003, the cited study reveals that players were unaware of loss rates of up to 5% while the addition of 75-100ms of latency decreases the number of kills up to 50% over common Internet latency ranges. On the other hand, for Warcraft III, an online multiplayer "Real-Time" Strategy game (TCP based), a cited study reveals that Internet latency ranging from hundreds of milliseconds to several seconds doesn't significantly affect user performances. Regarding Madden NFL, an eSport game, a cited study reveals that player performances were not noticeably affected until the latency reached 500ms. In [48], the authors studied the effect of network conditions on player departure behavior in the MMORPG game "ShenZhou Online". According to them, unfavorable network conditions do have an influence on the departure behavior. In addition, they found that when players quit in unfavorable network conditions, 10% of their dissatisfaction is due to network latency, 20% is due to jitter (defined here as the standard deviation of packet Round Trip Time - RTT), 40% to client packet loss (i.e. in the uplink direction), and 30% to server packet loss (i.e. in the downlink direction). These results should be analyzed knowing that the transport protocol used by "ShenZhou Online" was TCP, which means that packet loss at the network level translates into delay at the application level. For the average RTT, the authors identified a threshold effect at 180ms. Under this value, the average RTT has no impact on the departure behavior while it has a growing effect above this value. In [107], the authors focus on the prediction of network path quality between P2P players in the context of the Halo3 game (in a Halo3 game of up to 16 players, one of the player's console is selected as the game host that relays the traffic between the other players which can consume up to 1Mbps ((16-1)*70Kbps) of bandwidth). The authors consider that the gameplay is excellent when the latency (here the RTT) between players is under 50ms and that a "good" experience can be achieved with 150ms of latency. They study a large data set of measurements collected on the Xbox Live service in 2008 which covers 5.8 million unique IP addresses that played 39.8 million game sessions in 50 days. Their results give some information on the temporal and geographical correlations between the network path quality data collected by the Xbox Live service, and concludes that it is possible to predict which pairs of P2P games will provide a "good" experience.

2.2 Cloud gaming

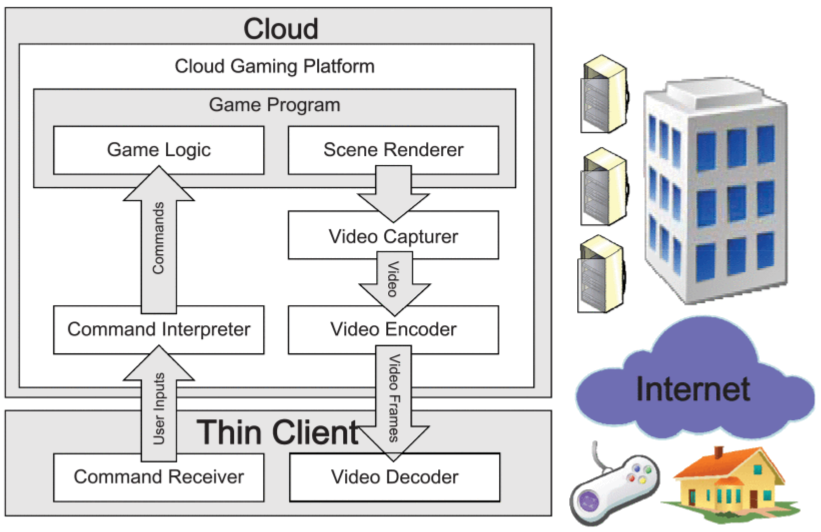


Figure 3: Typical cloud gaming services [44]

In cloud gaming, the player sends its inputs to a game executed in the cloud which in return produces video frames that are streamed back to the player. A typical cloud gaming architecture is represented on Figure 3. Similarly to online gaming, latency compensation techniques have specifically been developed for the cloud gaming context. Google markets its technology as "negative latency" (more details below). A research work contributed by Microsoft [105] recognizes a threshold of 100ms of network RTT above which many gamers complain. With their latency compensation techniques, called "Outatime", they claim to be able to mask up to 250ms of network latency.

[44] is a survey on cloud gaming published in 2016. It covers cloud gaming platforms, QoS and QoE evaluations, the optimization of cloud gaming platforms and commercial cloud gaming services. It lists several studies showing the sensitivity of cloud gaming to network latency. According to a study [151] that used a simplified game (push of a button triggering a simple visual presentation), half of the gamers cannot tolerate more than 100ms response delay (defined as the visual delay that follows user actions. As such, this includes, but is not limited to, network latency).

In a 2012 study [57], the authors analyzed the traffic characteristics of the OnLive cloud gaming platform. They found that OnLive used UDP with downstream bitrates about 5 Mbps and 1000 bytes packets while upstream bitrates were about 100Kb with 150 byte packets. More precisely they found the downstream bitrate to depend on the game with two games generating about 6.3Mbps and the third one 3.8Mbps. The upstream bitrate showed considerable variation over time for the three games with variations between 50 and 150Kbps, with one of the games (a fast action FPS) having slightly higher bitrates than the other two.

In [49], the minimum bandwidth requirements of 3Mbps and recommended bandwidth requirements of 5Mbps for OnLive services at 720p format are recalled. On the other hand, StreamMyGame software announced requirements range from 256Kbps for 320x240 resolution to 30Mbps for 1080p. The authors found that the average playout delay (defined as the difference between the time the client receives an encoded frame and the time it is decoded and displayed) is about 20-30ms for OnLive and 15-20ms for StreamMyGame while the average processing delay (defined as the difference between the time the server receives a player's command and the time it responds with a corresponding frame) is about 200ms to 100ms (depending on the game) for OnLive and about 400ms (also depending on the game) for StreamMyGame – the difference between is attributed to either hardware h264 encoders or more powerful CPUs at OnLive.

In [106], the authors conducted user studies, with 15 users, involving nine games in three different genres and they measured the players' emotions, using the electrical signals produced by a facial muscle expressing frown and suffering, when different levels of latency were introduced in the input-response loop. They showed that game cloud-friendliness (i.e. the latency introduced divided by the difference of QoE, as measured with facial EMG, without and with additional latency) depends on the game. Moreover they verified that cloud-friendliness is correlated with its command heaviness (in pixels per command), i.e. with the screen dynamics (a measurement of the quantities of pixels moved between each frames) divided by the command input rate. They also built a regression-based predictive model deriving the cloud friendliness of a game from its command heaviness.

In [121], the authors characterize the network traffic generated by two cloud gaming platforms: OnLive and Gaikai. The network requirements announced by OnLive were already mentioned earlier. Gaikai announced requirements are similar: it is said to be playable at 3Mbps but offers better performances at 5Mbps and can stream at 1080p with higher bandwidth needs. Results are presented in packets per seconds and packet sizes (as CDF – and not reported in this document). In the downstream direction, OnLive ranges between 600 to 750 packet/s while Gaikai ranges between 350 to 550 packet/s. In the upstream direction, OnLive ranges between 35 to 65 packets/s and Gaikai ranges between 10 to 40 packets/s. In a follow-up paper [122] dated 2014, the authors dissected the network protocols used by OnLive and revealed that both the downstream and upstream flows (conveying the mouse and keyboard commands) are RTP/UDP-based.

In [46], the authors speculate about the latency compensation techniques used by Google in Google's Stadia and marketed as "negative latency". According to them, these involve a combination of "closeness to the datacenter, predictive inputs, server side running at a higher frame rate for faster reactions and parallel sending of speculative frames (i.e. many possible actions at once). They also reveal that Stadia is WebRTC based. Similarly to the other platforms, the traffic characteristics depend on the game played. They found the average downlink throughput to be of 23.63Mbps and 2.07 Mbps respectively for Tomb Raider and Spitlings (a 2D game with slow moving graphics) when H.264 CODEC is being used while, when VP9 is being used, these values respectively become 27.56Mbps and 2.10 Mbps. The bandwidth requirements announced by Google are 10 Mbps to use the service at 720p, 28Mbps for 1080p and 35Mbps for 4K (2160p) which was found to be consistent with traffic analysis except for 4K where the traffic load was higher than 35Mbps 88% of the time. The authors tested Stadia's adaptation to bandwidth change (i.e. drop/increase to 10-20-30Mbps) and found that the transient behavior where Stadia's change resolutions until it converges to a stable one could be rather long, i.e. in tens of seconds. Playability is severely affected during such episodes. This paper also covers some latency analysis. However it is limited to the sole network access involved in the data collection, i.e. a fast access (100Mbps) with a low base latency. Network RTT values (measured using STUN round-trip-times) were consistently below 25ms and, in average, between 10 and 15ms. The (adaptive) jitter buffer, which is used at the client side to smooth the playout of video frame was found to contain up to 2-3 video frames which represent 35.76ms and 45.35ms at 60fps. Thus, the button-to-pixel latency is below the 60ms where users can start noticing latency issues [105].

In [123], also analyze the data consumptions of cloud gaming and contrast it with gaming from self-owned devices (console, PC, mobile). The average throughput they found for Google's Stadia with the game Metros Exodus (a single player game) is consistent with other studies, i.e. 10.06Mbps at 720, 27.24Mbps at 1080p, 41.54Mbps at 4K when played from a PC or console and 10.4 Mbps from a mobile (720p). They also tested the NVIDIA GeForce Now offer with the game Counter Strike at 1080p and found an average of 11.9Mbps when played from a PC and 6.6Mbps when played from a Mobile (over Wi-Fi).

In [60], the authors investigated how user experience changes under sub-optimal network conditions with two recent cloud gaming platforms: Google's Stadia and Blade's Shadow. This investigation was conducted with 36 users and a selection of various games. Both objective and subjective data were collected. The test was conducted by adding the following latencies of: 0ms, 50ms, 100ms, 200ms and 500ms. The following inbound and outbound packet loss conditions were also tested: 0%, 10%, 20%, 40%. Shadow had about 3-4 times higher bitrate than Stadia in most trials and in general Shadow's bandwidth consumption stayed relatively constant while Stadia varied depending on the added delay and packet loss. Their results show an almost linear decrease in perceived responsiveness as latency increases (the addition of 50ms of latency already showing a decrease from 0ms) and a willingness to play sharply decreasing between 100ms and 200ms of added latency. The network access used for the tests was from Worcester Polytechnic Institute LAN. Regretfully, it has not been further characterized in the study. The impact of latency on player's performance during the game (i.e. number of kills and number of time the player took damages) also show a trend toward worse performance as the latency increases but the data is noisier than the subjective rating.

In [182] , the authors study adaptation mechanisms under varying network conditions that are implemented in the cloud gaming platform NVIDIA GeForce NOW (GFN) which also used RTP over UDP. They analyze the effect of network delay, delay variation, packet loss and bandwidth shaping using three different games and found the bandwidth adaptation mechanism to be dependent on delay but not on packet loss. The characteristics of their test network as estimated by the GFN platform were: above 50Mbps, no packet loss, latency of 22ms and jitter of 18ms. In these condition, bandwidth usage by the platform is around 30Mbps and videos are displayed in 1080p at 60FPS. They notably discovered that the effect of adding a latency of 10ms to the nominal network latency was different whether it was added before gameplay (GFN has can estimate network conditions before playing to set the adequate resolution and framerate) or during gameplay. When latency is added during gameplay, the platform quickly drops is throughput to 2Mbps and then takes approximately 2 minutes to adapt. When the added latency is removed, the platform only takes seconds to recover its nominal settings (around 30Mbps with a video of 1080p@60FPS). They also ran QoE experiments with 15 adults in various network conditions. The QoE is significantly lower when additional latency (20ms) is added during gameplay and not very much impacted if added before. Packet loss (they tested 3%, 5% and 10%) had a very limited influence on the QoE.

2.3 Live video streaming

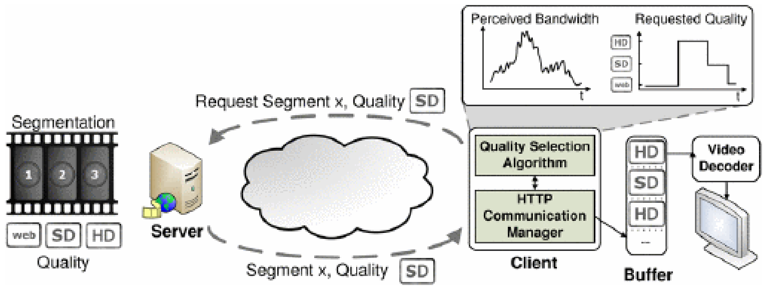


Figure 4: Schematic overview of the HTTP Adaptive Streaming (HAS) concept [56]

Video streaming over HTTP is nowadays usually delivered using an adaptive bitrate streaming technique such as in MPEG-DASH (Dynamic Adaptive Streaming over HTTP), Apple's HLS (HTTP Live Streaming) or Microsoft's Smooth Streaming. In these techniques, the video content is encoded at multiple bitrates and the client selects the appropriate video bitrate depending on the network conditions. These video contents are segmented into subparts which are typically between 2s and 10s in length. Doing this intrinsically generates a high glass-to-glass (i.e. between content generation and its consumption) latency of tens of seconds due to the duration of the segments and the needed buffering of several segments at the media player to achieve a smooth playback. Several proposals exist to reduce this latency and make it more comparable with other video broadcasting technologies (e.g. DVB) such as ALHLS (Apple's low-latency HLS) and the Chunked CMAF for MPEG-DASH (Chunked Common Media Application Format) also known as low-latency CMAF which aims to achieve sub-second glass-to-glass latency. In chunked CMAF, a video segment is split into several shorter chunks, also known as fragments, which include the minimum data required to start decoding the stream and, in turn, allow prompt playback start as the full segment does not need to be downloaded to start the playback.

The QoE metrics for HTTP streaming receivers, whether for on-demand streaming or for live streaming, have been well studied in the literature. In [185], the authors evaluate the effect of varying the chunk (fragment) duration on the resulting latency and on the user's QoE. They note that evaluating the resulting latency alone is not sufficient as diminishing chunk duration can have a positive QoE impact on the initial delay but also a negative QoE impact on the stalling time and the number of representation switches. According to the authors, the network latency can be neglected in the formula used to calculate the end-to-end latency. In their experiments, they tuned the buffer size of the player to be equal to one fragment duration and test different configurations with 1, 3, 5 and 60 frames per fragment (assuming a frame rate of 30 fps this means fragment duration of 33ms, 100ms, 166ms and 2s), with a player connected to a Wi-Fi access point. As expected they observed increasing end-to-end latency as the number of frames per segment increases (resp. around 120ms, 190ms, 260ms and 2200ms). They also observed that the number of freezes and the average freeze duration increase when the number of frames per fragment (thus also the player buffer size) diminishes. In their experiments, the capacity and latency variations of the wireless link is not traced. Moreover, the authors do not establish a link between these aspects and the playout freezes, and only look for solutions in the configuration of Chunked CMAF.

In [180], the authors propose a live video streaming algorithms to maximize user QoE. In the design of the algorithms they took into account the option of streaming video using chunk-based streaming. They considered chunks of 200ms. Using a trace driven simulation based on a real 4G cellular bandwidth dataset collected in the NYC metro area, they shown the possibility to lower the latency in the range of 2 to 5 seconds with an average video freeze duration below 1s, but also some percentage of cases with more than 4s of freeze time for a 100s test duration.

In [21], the authors notice that bandwidth estimation based on the download duration of a segment doesn't work in Chunked CMAF (due to the presence of idle periods between chunks constituting a segment). This is of course an issue for the rate adaptation heuristics of a HAS player. In an experiment using Twitch with low-latency mode enabled, they artificially varied the network bandwidth (between 1-5Mbps) between the player and Twitch server. On Twitch, the video of the live channel was encoded at six bitrate levels of 0.18, 0.73, 1.83, 2.5, 3.1, 8.8 Mbps with three resolutions of 540p, 720p, 1080p. This revealed that the player was unable to adapt to varying conditions and kept on selecting a video bitrate of 1.83Mbps. Thus, they propose a chunk-aware bandwidth estimation method which is also complemented by a bandwidth prediction module and a controller to select the bitrate of the video segments. To evaluate their proposal, they did some experiments in a controlled environment with arbitrary bandwidth variation profiles using chunk duration of 500ms and a target latency of 3.2 seconds which values are based on an existing live service.

In [86], the authors propose an HTTP Adaptive Streaming approach for live video streaming that is based on the (server) push feature of HTTP/2 [4]. They highlight that at least one extra round-trip-time is lost between each video segment request which can be significant in high RTT networks and when the video segment duration is short (i.e. sub-second). In a previous proposal, the authors assumed that pushed segments are delivered within a certain time interval. As this does not hold in capacity varying mobile networks they propose an improved approach which consist in limiting the number of spontaneously pushed segments depending on the RTT and segment duration. They experimented their proposal with a 3G/HSPA network trace (where the average RTT was around 200-500ms and the throughput between 500kbps-5Mbps) and a soccer game video of 596s. They report achieving a total freeze duration of 1.38s, a startup time of 1.25s and a server-to-display delay of 8.26s (noting that this doesn't take into account the time necessary to encode a live video). These results are obtained with segment duration of 500ms and a buffer size of 6 segments, and were obtained without using chunked CMAF.

In [77], the authors focus on user generated 4K HTTP live streaming when receivers are connected over an LTE-A access with a high RTT network connecting to the video source. Due to TCP inefficiencies, in these conditions one obtains a poor QoE. The authors consider that traditional CDN is a solution to this problem but there are common cases where there are cache-misses so the receiver connects directly to the video source and the problem of TCP inefficiencies arise again. They propose a solution relying on a virtual edge cache (i.e. a modified HTTP Proxy) deployed at the MNO by the CDN operator in which the cache is holding back some segments at the receiver. The challenge they solve is the definition of segment holding policies (at the edge cache) depending on: the backhaul (e2e latency), the video bitrate and segment length and the radio signal quality. Since they focus on 4K video, the bitrates considered are 15Mbps and 50Mbps. However, in their experiments, they only considered relatively long video segments of 2s, 4s and 10s. They provide experimental results in many different configurations and claim to reduce the total live latency as compared to an end-to-end solution with no such edge caching.

2.4 Conversational services

Audio and audio-video communication between humans is known to be sensitive to the network transmission delay for a long time, even since the days of circuit switched networks. In [76], the ITU-T defines acceptable transmission delay at about 150-200ms with 400ms as an absolute maximum. The "E-model" [75], a computational model, was used to derive the impact of delay on the "rating" of the transmission quality: a score, which can be converted to a mean opinion score (MOS) regarding call quality. According to [76], if all other parameters governing voice call quality were perfect, the relation between the rating of the transmission quality and the one-way delay would be given by Figure 5 showing the effects of

[4]Editor note: it seems the author transposed to HTTP/2 the way RTMP works (i.e. with a push model) – see also https://engineering.fb.com/2015/12/03/ios/under-the-hood-broadcasting-live-video-to-millions/

absolute delay as defined by the E-model [76].

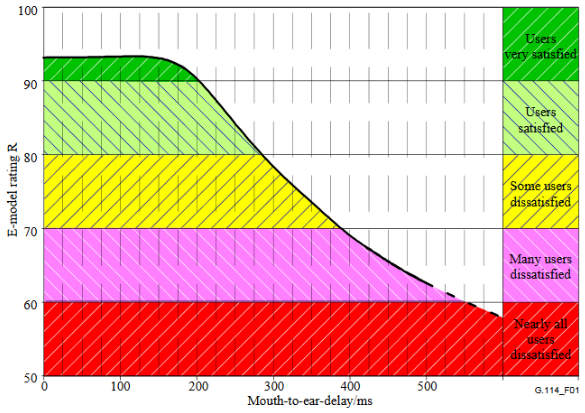


Figure 5: Determination of the effects of absolute delay by the E-model [76]

In [92], the authors survey VoIP quality of experience assessment approaches. They identified [136], in which the authors studied the trade-off between latency and packet loss. Figure 6 shows the perceived quality contours in the case where there is no echo and the CODEC used is G.711 with Packet Loss Concealment (PLC).

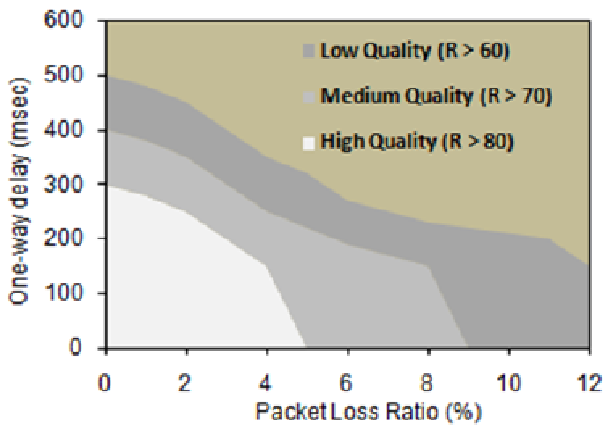


Figure 6: Users' satisfaction for G.711 CODEC with PLC

In [191], the authors study the design choices and the performances of three representative video telephony applications, namely Google+ (now Hangout), Apple iChat (now Facetime) and Skype. The studied design choices cover the system architecture (peer-to-peer vs relayed), the video generation and adaptation (simultaneous use of multiple versions of video quality), the packet loss recovery (forward error correction vs retransmission) and the adaptation to network conditions. They discovered that the frame rate could vary from 1 frame-per-second (FPS) to 30 FPS. Resolutions varied between 160x120 to 640x480 and the bitrates varied between 5 kbps to 1200 kbps (these figures aggregate voice and video bitrates). Although, the bandwidth consumed by a video stream can be adjusted with its quantization, this parameter could not be directly measured with the considered testbed. Regarding the multiple versions of a video stream, the authors discovered that iChat used one-version encoding, Skype used multi-version encoding (up to three) and Google+ used multi-layer encoding using both temporal and spatial scalability (with two spatial layers). The authors also measured the one-way voice and video end-to-end delays (i.e. taking into account endpoint processing time such as encoding and decoding). For Skype, with a negligible network delay, they measured a one-way video delay of 156ms and a one-way voice delay of 110ms showing that the endpoint processing delay is not negligible in the end-to-end delay. Using a similar methodology, Orange measured in 2015, the one way end-to-end video delay with two smartphones using 4G for Skype, Hangout and ViLTE. The results are summarized in Figure 8.

In [91], the authors recall the range of possible bitrates for the opus audio CODEC: from 6kbps to

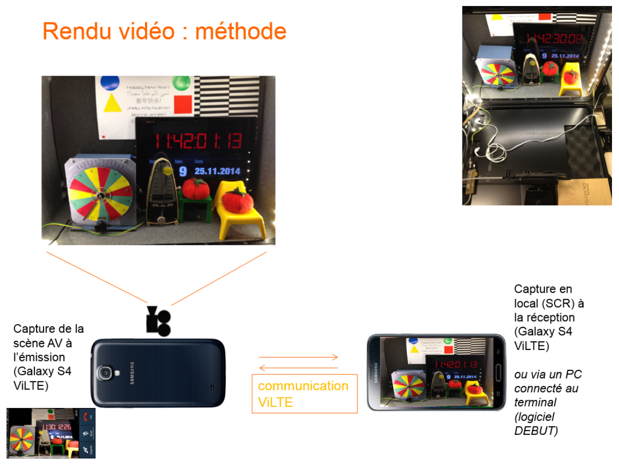
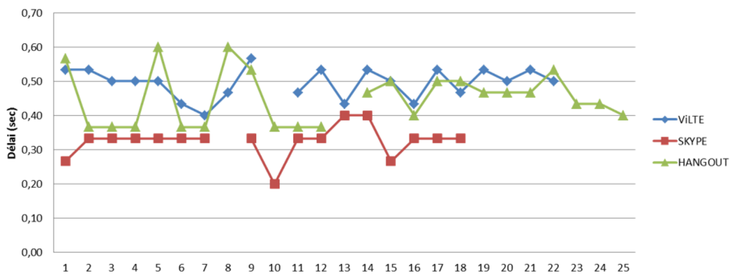

Rendu vidéo : méthode
Capture de la scène AV à l'émission (Galaxy S4 ViLTE)
communication ViLTE
Capture en local (SCR) à la réception (Galaxy S4 ViLTE)
ou via un PC connecté au terminal (logiciel DEBUT)

Figure 7: Video rendering



Service	ViLTE	SKYPE	HANGOUT
Délai moyen	0,50 s	0,33 s	0,45 s

Figure 8: End-to-end one-way video delay of ViLTE, Skype, Hangout over 4G

510kbps (for high quality stereo music). The VP8 video CODEC bitrate is also given depending on the resolution and frames per seconds: from 1-2Mbps at 720p and 30 FPS; 0.5-1Mbps at 360p and 30FPS and 0.1-0.5Mbps at 180p and 30 FPS. To handle packet loss for VP8 video flows, WebRTC implements a hybrid approach combining forward error correction (FEC) and negative acknowledgment (NACK) based retransmission: if RTT is low, packets can be retransmitted and FEC is set to a low level of redundancy, while if RTT is high, FEC redundancy is higher. It is also pointed out that WebRTC has an adaptive playback delay [5] which allows to modify the playback delay while waiting for packet retransmission, thus reducing the duration of frozen video.

While [40] deals with TV broadcasting, its recommendation that synchronization between audio and video components must be maintained would also apply to audio-video conferencing. According to [40], a lag of 45ms to 125ms becomes noticeable while a lag of 90ms to 185ms becomes unacceptable.

The IETF (Internet Engineering Task Force) document "Evaluating Congestion Control for Interactive Real-time Media" [174] defines network scenarios recommended to be used for evaluating the performances of congestion control for interactive point-to-point real-time media. These scenarios cover various configuration regarding:

- One-way propagation delay: Very low latency: 0-1ms; low latency: 50ms; High latency: 150ms; Extreme latency: 300ms
- End-to-end loss: No-loss; 0%; 1%; 5%; 10%; 20%
- Loss generation models: Independent random losses at minimum
- Queuing policy: Drop-tail at minimum

[5]See also https://webrtc.googlesource.com/src/+/refs/heads/master/docs/native-code/rtp-hdrext/playout-delay

- Queue length: Short: 70ms; Nominal: 300-500ms; Buffer-bloated: 1000-2000ms
- Jitter models: Two jitter models are defined in the document: Random Bounded Packet Delay Variations (RBPV) and Approximately Random Subject to No-Reordering Bounded Packet Delay Variations (NR-RPDV)
- Traffic models for TCP, RTP video and background UDP

2.5 Drone piloting

According to [181] aerial drones are a special type of Unmanned Aircraft Vehicles (UAVs) which are either autonomous or remotely controlled (on the contrary of some other UAVs, such as missiles, that may be uncontrolled). Drones can be directly controlled by humans, by computer programs or a combination of both. In the case the drone is controlled by a human, we can distinguish the case where the drone is visually controlled by the operator from the ground (also known as Third Person View), and the First Person View (FPV) case where the pilot is relying on a video stream from an on boarded camera to control the drone. In the case of a computer controlled drone, the drone can be autonomous, i.e. the needed computing resources are on boarded, controlled from computing resources that are remote from the drone or a combination of both. Here, we are not interested in the fully autonomous case as it involves no networking. We are more interested in the scenarios where the on boarded computing resources are constrained (e.g. for physical or cost optimization reasons) as it maximizes the networking requirements. A specific scenario that apparently maximizes network requirements is the case of a drone swarm (or multiple coordinated drones evolving in close proximity) with a centralized control.

2.5.1 Human drone piloting

Drones can be controlled either from a third person view (TPV) or from a first person view (FPV). According to [175], TPV operation creates control and safety problems as operators have difficulties in determining the drone's orientation and keeping eye contact at beyond relatively short distances (>100m). When the article was published in 2017, First Person View (FPV) drone systems transmitted the video feeds over analog signals allowing for low-latency video transmission and graceful reception degradation.

In the marketing product document [115], an example of end-to-end latency from the video capture (on the UAV) to the video display (on the pilot side) is analyzed in order to compare two video encoding techniques: one where there is only a single slice per frame and the other with multiple slices (30 in the example taken) per video frame. Slices are independent video encoding/decoding units. Having multiple slices per video frame allow parallelizing the encoding/decoding and transmission thus reducing end-to-end delays. In the example, the UAV is connected to the pilot remote control over Wi-Fi with 2Mbps of bandwidth and a video stream of 1Mbps. A direct Wi-Fi link is assumed (no network). With one slice per frame the theoretical end-to-end latency is 118.8ms while with 30 slices per frame, the theoretical end-to-end latency is 39.9ms.

According to the authors of [115], drone racing championship winners can fly at average speeds of 30m/s (i.e. approx. 110km/h). The authors advocate for the relevance of stereoscopic drone operations for FPV control but also for autonomous control as this would allow the autonomic system to acquire depth information and to estimate the scale of the surrounding objects. They designed such a system working with H264 CODECs. Rendering is done at 75 frames per second which allows panning (i.e. the operator changing the HMD field of view due to head movements) limiting motion sickness. Video streaming was performed on UDP over Wi-Fi. The total system latency is about 250 +/- 30ms from the action execution until the video goes through the HMD (aka button-to-pixel latency). This could be improved with more powerful graphic cards for the encoding process. In the experimentation with human pilots, the pilots were able to achieve speeds of 7 m/s with no accident. While controlling the drone, the pilots could not notice the latency. Apparently, 7 m/s was the upper limit after which crashes were possible.

2.5.2 Automated drone control

The maximum latency a robot can tolerate to avoid collisions is related to its speed, the range of its sensors and its actuation limitation (e.g. the acceleration it can produce) as shown in [67]. In this work, perception (i.e. sensing) and actuation limitations are jointly considered. The authors developed a mathematical model and applied it to the case of an autonomous UAV sensing its environment with different types of cameras (monocular frame-base, stereo frame-based, event-base) having different latency and sensing range characteristics. Interestingly, they applied their model to compute the maximum sensing pipeline latency that a robot can tolerate to avoid collisions depending on its speed, in the cases of an acceleration capability of 20m/s2, 50m/s2 with a sensing range of 2m, 5m.

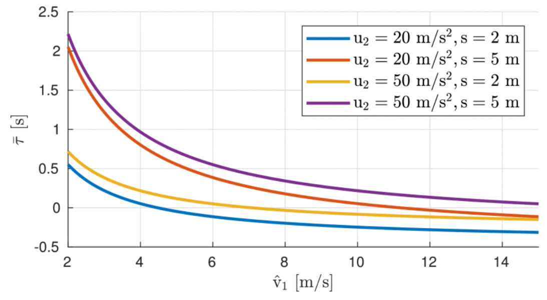


Figure 9: Maximum latency to perform avoidance maneuver with an obstacle of 1m of projected length [67]

The authors of [67] also studied the constraints of current commercial camera systems and current commercial quadrotor platforms. The sensing range of the different type of cameras vary between 2m and 8m while the latency of the related sensing pipelines vary between 2ms and 70ms. Based on real commercial drone performances, they considered 3 (lateral) acceleration capabilities plus 1 beyond current performances (but considered achievable in some years): $10m/s^2$, $25m/s^2$, $50m/s^2$ and $200m/s^2$. The computed maximum longitudinal speed of the drone range from 3m/s up to 31m/s when current drone performance are considered and even 62m/s with an acceleration capability of $200m/s^2$. They notice that current racing FPV quadrotors are already capable of reaching speeds above 40m/s with thrust-to-weight ratios above 10.

In [28], the authors compare the latency of the proprietary protocol stack, working at 2.4GHz, normally used by Bitcraze's Carazyflie nano-drone with an open protocol stack based on Wi-Fi radio also working at 2.4GHz, IP, UDP and the Predictably Reliable Real-time Transport (PRRT) [27]. To control latency, PRRT employs cross-layer pacing and adaptive hybrid automated repeat request (AHARQ) to adapt the redundancy to the current channel characteristics and application requirements. PRRT's *send*() primitive allows to delay the sending for a relevant duration given the current bottleneck and the *recv*() primitive controls the treatment of packet expiry dates upon reception. On the drone-side, the open software protocol stack is implemented in two fashion: one using python, the other using rust. In both the cases of the proprietary stack and of the open stack, the same application layer protocol, the Crazy RealTime Protocol (CRPT), is used to encapsulate all control and feedback messages. In the evaluation scenarios, the drone is controlled by a PC over a 2.4GHz radio link. The drone streams the video of its onboard camera (at a target bitrate of 1.7Mbps) on the same radio link also used to control the drone. They found the RTT of the proprietary protocol stack to be about 4ms while the RTT of the PRRT-based stack implemented in rust is about 9ms and the python implementation is about 18ms. The authors do not attempt to identify specific latency requirements. On the opposite, they claim that the obtained worst RTT (with the python-based open stack) is acceptable for control scenarios that can work with 20ms. According to them, drone (in this case a UAV) demand predictable latency communication to maintain the stability of the physical process under control, i.e. to achieve a stable flight while following the desired path as close as possible or being at the correct relative position with regards to other UAVs in a swarm [7]. According to [148] such a control loop is evidently delay bounded. The time limit is mainly affected by the capabilities of the drone and environmental conditions as well. Having a fast drone or a quickly changing environment requires quick control decisions and low end-to-end latency. When the control equipment is close to the actual moving hardware (i.e. on boarded on the drone), the requirements are more easily satisfied.

In [133] , the authors give an overview of 3GPP's Release 15 study on enhancing LTE for supporting connected drones (i.e. on 3GPP's TR 36.777 [1]). The key issue addressed by this study is about interference mitigation as given their altitudes UAV will frequently have multiple base station in line of sight (LOS). Therefore uplink transmission to one base station has increased chances to interfere with the other base stations compared to ground-level UEs and conversely downlink transmission to one UAV has increased chances to interfere with other UAV. In this study, the following traffic model

[7]Editor Note: drone swarming can be a good rationale for optimizing control decisions by centralization rather than distributing it to on-boarded drone controls

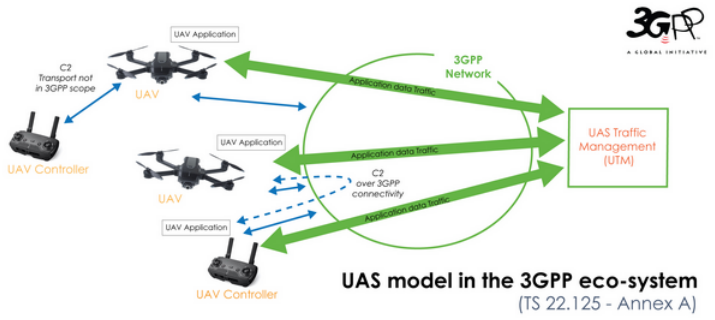

UAV Application
C2 Transport not in 3GPP scope
UAV
UAV Controller
3GPP Network
Application data Traffic
UAS Traffic Management (UTM)
C2 over 3GPP connectivity
UAS model in the 3GPP eco-system
(TS 22.125 - Annex A)

Figure 10: UAS model in the 3GPP eco-system - https://www.3gpp.org/uas-uav

is considered for UAV: the uplink/downlink data rate required for the Command & Control is about 60-100Kbps, the uplink data rate for application data goes up to 50Mbps, the maximum latency bound [8] is evaluated at 50ms for the Command & Control (C&C). The study [1] also defines different scenarios in terms of network density, UAV density, flying altitude and channel models. Besides providing data on the interference problem, the study also sheds light on the mobility performances of UAV UE [66] with potentially high radio link failure rates and handover failure rates in dense scenarios.
When drone automation is not on-boarded, automated drone piloting can be seen as a kind of cloud robotics application [94]. As will be further elaborated in Section 2.8, cloud robotics requirements are different depending on what is in the robot (e.g. a drone) and what is in the cloud. For example, in the case of centralized trajectory planning in the cloud and short-term decisions and collision avoidance in the robot: latency in the order of 100ms is sufficient [6].

2.6 Tactile Internet

Telerobotics can be classified as a tactile internet application. The tactile internet allows a human-machine interaction, in order to manipulate objects or devices with high reliability, very short transmission time and high level of security.

[69] is a vision paper published in 2014 advocating the emergence of the tactile Internet. According to the author, this would be triggered by the placement of exa-scale compute power in every access point or base station of future wireless systems. For the author, such exa-scale compute system would be enabled by the advent of 3D chip stacks. He also presents the characteristics of four types of physiological "latencies": muscular, audio, visual and tactile. Muscular reaction times (e.g. reacting to a fly settling on one of our limbs) is typically a second. When listening to voice and reacting within a conversation a latency of up to 100ms is non-observable. When watching a move, a frame rate of 100Hz is acceptable: this relates to a latency of 10ms between frames. When tactically steering or controlling an object, a reaction latency of less than 10ms is needed. The author takes the example of a finger moving on a screen at 1 meter per second: the reaction time of the screen must be 1ms to achieve an unnoticeable displacement of 1mm between the finger and the object to be moved. The author then envisages several application scenarios in the domain of health care, education and sports, car traffic, robotics and manufacturing, free-viewpoint video and smart-grid. However, in this paper the reasons why cellular network latencies below 10ms would be needed are only vaguely given, the characterization of the network latency constraints are left unspecified and the vision paper do not consider any alternative implementations not requiring very low-latency communications. To implement such network latencies the identified challenges were: the LTE frame structure (LTE OFDM symbols last approximately 70µs while the author estimate that packet transmission should not exceed 33µs); and what would *"look more like a circuit-switching challenge in*

[8]Here, the term latency is used as defined in clause 7.5 (User Plance Latency) of 3GPP TS 38.913: *"The time it takes to successfully deliver an application layer packet/message from the radio protocol layer 2/3 SDU ingress point to the radio protocol layer 2/3 SDU egress point via the radio interface in both uplink and downlink directions, where neither device nor Base Station reception is restricted by DRX"*. Editor note: Therefore it is a one-way delay measured at the RLC to MAC interface of the sender and the MAC to RLC interface of the receiver. Consequently, queuing delays in the RLC and upper layers are not included. This KPI objective of 50ms for C&C of UAV can be contrasted with the target value of 4ms for eMBB user plane latency

[6]In the case of the centralized intelligence of a pool of machines (sensors and actuators): information from the machines (in the uplink) would need to be communicated at a high frequency (order of magnitude: 1ms), commands from the cloud (in the downlink) would be much less frequent.

terms of latency and packet reservation guarantees than today's mainstream packet switching paradigm". The author also identifies that for the application domains that would benefit from very low latency, high reliability would also generally be needed. He sets a target availability of seven nines (i.e. failure rates of 10-7) that can correspond to an annual outage of 3.17s. The avenues for attaining such a reliability is identified as being transmission diversity (e.g. MIMO, carrier-aggregation and coordinated multipoint). Finally, the author explain why 1ms latency implies placing powerful servers at the base station: at the speed of light a round-trip interaction involves that the client and server are less than 150km away (150kmx2 = 300km which is the distance travelled by light in 1ms). According to him (and without further explanations), taking signal processing, protocol handling and switching delays into account this would only leave 15km between the tactile point of interaction (which is approximately the radius of large cells in a cellular network).

In a more recent work published in 2017, [9], the authors review in more depth the design challenges of haptic communications and propose first avenues for specific solutions. According to the authors, the tactile Internet is a paradigm shift from content-delivery networks to skillset and labor-delivery networks. The tactile Internet provides the medium for transporting touch and actuation in real-time. They recall that state-of-the-art 4G cellular networks have a latency in the order of 20ms [1] while 5G aims at supporting 1ms of RTT. They distinguish two distinct types of information constituting haptic feedback: kinesthetic feedback which provides the information of force, torque, position, velocity… and tactile feedback which provide information about surface, friction…. The body sensors are different in the two cases (muscles, joints… in the former and human skin in the later). Only kinesthetic information closes a global control loop with stringent latency constraints. They clarify the tactile Internet is not only for humans as it enables networked control systems with sensors and actuators involved in highly dynamic processes, however they focused on cases involving at least one human user. They identified many challenges for the tactile Internet among which: the development of haptic devices (the few that are commercially available are bulky and expensive; they mostly manage single points of contacts and not whole body areas), the need of standard haptic CODECs and haptic data compression (haptic signals being typically sampled at 1 kHz this would lead to 1000 packets per seconds – compression of haptic data would exploit the limits of human haptic perception to send less data through bandwidth-limited networks), the need to deal with multi-modal sensory information, i.e. haptic combined with audio-visual (existing efforts assume perfect constant bitrate channels), the stability of haptic control considering the time-varying nature of networks (the propose ML-based predictions at the network edge), ultra-reliability, ultra-responsive connectivity (they re-state the requirement of 1ms of round-trip latency and specifically point to the need of shorter OFDM symbols and shorter TTI), radio resource allocation (the specifically point to the need of semi-permanent allocation in the uplink – downlink), multi-user haptic communications (they point to the management of the decentralization and co-ordination between peers) and the lack of objective quality metrics for haptic communications. The network overhead in the transmission of haptic data is also mentioned by the authors: haptic data is typically sampled with a resolution of 16 bit per Degree of Freedom. A 3-DoF haptic stream can be of only 6 bytes per packet: with such small data the encapsulation overhead can be huge (e.g. the size of IP/RTP/UDP headers is 40 bytes) and require the use of header compression techniques[2].

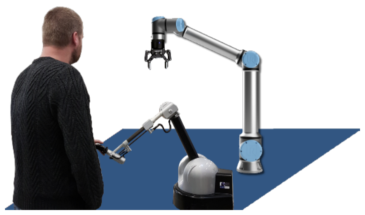


Figure 11: Example of haptic device, the 1to1Telerobotics extender from Haption and Universal Robotics

2.7 Medical telerobotics

The study[15] is a systematic review of the literature dealing with teleoperated medical robotic systems between 2004 and 2015. The authors distinguish between short distance (typically between two rooms

[1]Editor note: this figure is only valid for sporadic traffic in favorable conditions

[2]Editor note: the throughput of the haptic stream alone is however quite limited: 48 Kbps for the payload and 368 Kbps for IP

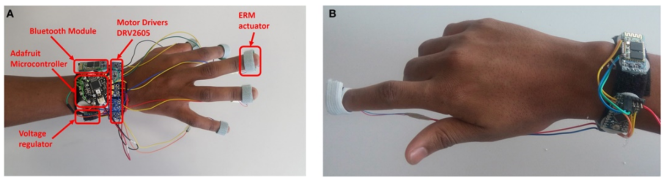


Figure 12: An experimental wearable vibrotactile haptic device [120] : (A) in five finger setup and (B) single finger setup

in the same location, mainly driven by the need of a physical barrier) and long distance telerobotics. In telerobotics, the local and remote systems are typically referred to as “master” and “salve” systems. In Medical telerobotics these are typically the “expert” and “patient” sites. They recall that the first successful telesurgery named “operation Lindbergh” occurred in 2001 with a patient in Strasbourg and a surgeon in New York. It used a dedicated optic fiber through the Atlantic Ocean. The control of telerobotic systems is primarily based on image (imaging method can be specific to laparoscopy, ultrasound, computed tomography, magnetic resonance imaging and X-ray fluoroscopy) and video guidance. However, force-reflecting haptic feedback is mentioned as a breakthrough facilitating telemanipulation. Auxiliary control functions found in medical robotics include motion scaling, biomotion compensation (e.g. follow the heart motion during an intervention) and hand-tremor filtering. In a telerobotic system, the master station controls a remote robot by sending position commands and accepting force and visual feedback, in addition to information on slave robot position and status. They distinguish between the following three types of dataflow: real-time control data which is bi-directional when there is force-feedback (at constant rate and constituted of small packets), medical video stream, high-level management data. The network issues reported in this study are the same as in [9]: current solutions assume a constant network delay. They also assume that passive forces are applied (by humans and the environment) to master and slave robots (which is a simplification). No solution deals with the instability of closed-loop haptic control in presence of varying latency, jitter, packet loss and throughput. Video compression needs to be diagnostically-driven, i.e. improving the diagnostic capacity of the medical video, the prevailing approach being the region of interest encoding technique. Robot control shall be prioritized over medical video typically using multi-objective optimization.

Figure 13: The da Vinci robotic system[15], in its short-distance version

In the MELODY system landmark example, it is noted that the robot is open-loop-controlled over the communication link but locally closed-loop-controlled on the patient site which removed the control instability problem (this idea was also used in the Robot in Medical Environment project). Again, in this context the literature assumes an update rate of the robot control data of up to 1kHz without loss or jitter to render the teleoperation “transparent”. It is noted that in this application there are robot control data, video-conference data and ultrasound video. The communication link between the expert and patient site require at least 256 Kbps. The number of telerobotic systems targeting short-distance scenarios is much larger than the one targeting long-distance scenarios: 45 references (35 experimental, 4 commercial, 6 clinical) vs. 9 references (8 experimental, 1 commercial: the MELODY system) in[15]. The medical domains addressed in long-distance scenarios are: general surgery (3 cases), spinal intervention (1 case) and tele-echography (5 cases). The network requirement vary a lot between the reported telerobotic systems. For example, in the case of the telerobotic ultrasonography experiment conducted by Sengupta et al. broadband Internet connections of 100Mbps and 50Mbps (for the Intercity and Trans-Atlantic

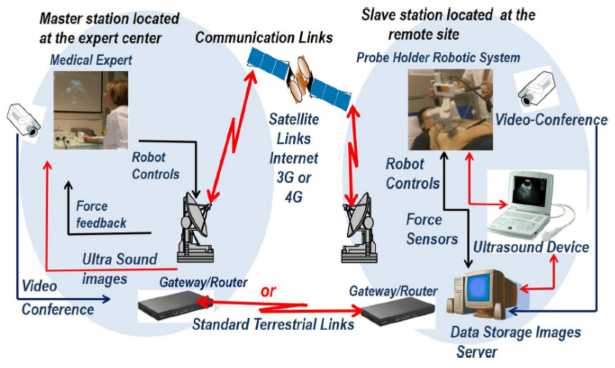

Master station located at the expert center
Medical Expert
Communication Links
Slave station located at the remote site
Probe Holder Robotic System
Video-Conference
Robot Controls
Force feedback
Satellite Links Internet 3G or 4G
Robot Controls
Force Sensors
Ultrasound Device
Ultra Sound images
Video Conference
Gateway/Router
or
Gateway/Router
Standard Terrestrial Links
Data Storage Images Server

Figure 14: Description of the robotized tele-ultrasound using the MELODY system[15]

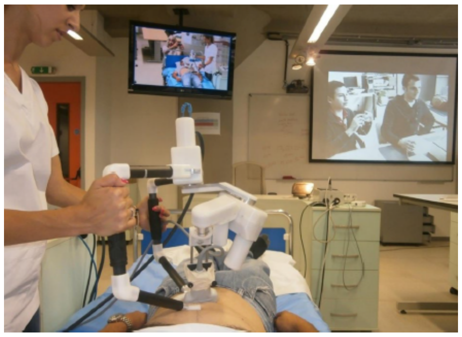


Figure 15: The robotized tele-ultrasound using the MELODY system[15]

cases) was used and occupied by the video feeds of a dual camera system, sound feeds and the real-time ultrasound images. On the expert site, the control interface was composed of a conventional mouse, a keyboard and the dials of the remote control interface (thus, no force-feedback is considered in this experiment).

2.8 Cloud robotics

Since several years, the evolution of robotics, artificial intelligence and cooperative reasoning has led to the deployment of robotics in several contexts in which the human presence is either not desirable (e.g. disaster situation) or avoidable (i.e. industry 4.0). If an initial vision of robot deployment consists in a standalone usage, it appears that the coordination of clusters of robots brings a relevant value-added service.

In this context, one needs to enable the robots to communicate through adapted wireless means and implement distributed algorithms allowing them to cooperate to accomplish their tasks. Such a processing can be implemented based on a peer-to-peer model. This allows robots to directly cooperate and avoid the need for any central infrastructure [50], or through a centralized component that can foster all the data related to robots but also their environment. Particularly, cloud architectures have been considered for dealing with data volume, processing complexity which consequently also enables the simplification, cost-reduction and energy-saving of robots that only conserve their fundamental capabilities to act and communicate. As depicted in Figure 16, cloud-based architectures, also known as cloud-robotics [93], are promising solutions to further develop clusters of robots, whatever their location. However, they also introduce a set of challenges, such as the latency due to the data exchanges between the robots and the cloud.

Two cloud models are considered in the literature for computation offloading. On the one hand, private clouds, with resources that are dedicated to the sole robot control, are often deployed directly on the site where robots operate. As such, only a limited latency and jitter are introduced, leading to a more stable and potentially satisfying response time. On the other hand, public clouds are also considered due to their low-costs not requiring the implementation of a dedicated infrastructure. However, in this case, since both the communication and processing are performed over mutualized infrastructures, ensuring the quality of services is challenging. An attempt to build a low-cost global system composed of commodity sensors and a supervisor control deployed in a public cloud [132] demonstrates that, given a probing/control

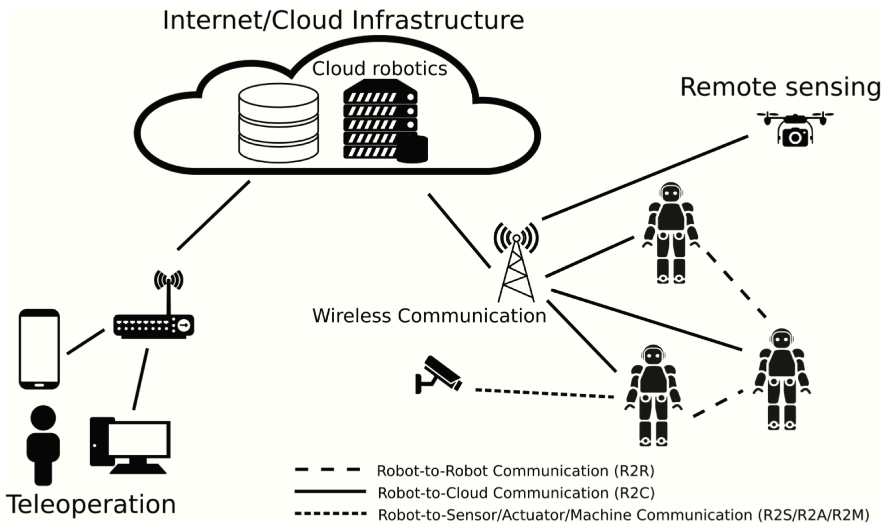

Internet/Cloud Infrastructure
Cloud robotics
Remote sensing
Wireless Communication
Teleoperation
Robot-to-Robot Communication (R2R)
Robot-to-Cloud Communication (R2C)
Robot-to-Sensor/Actuator/Machine Communication (R2S/R2A/R2M)

Figure 16: Overview of a cloud-based robotics architecture [19]

frequency of several hundreds of Hertz and using UDP as a transport protocol, the response time of the system is in the average 43.7ms while the sole communication part induces from 33ms to 42ms according to the packet sizes. This illustrates that an implementation with components taken off-the-shelf can provide a first satisfying experience. Some realistic response times collected from a client/server automation deployed over an Ethernet switch are provided by [5] whose purpose is to propose a response time estimation model. Such a scenario does not explicitly rely on a private cloud infrastructure, but one can reasonably expect a comparable behavior. Here the measured response time varies between 10 and 22ms, thus demonstrating the impact of remote vs. local supervisor/control infrastructure. The most stressful situation is experienced in [114] which considers the case of humanoids robots controlled by a cloud whose implementation is neither local nor geographically far. A relevant point of this study relies in the consideration of several control loops, all requiring different frequencies and, thus, response times. The most stressing one addresses position, velocity and torque control with a probing frequency of 1 to 5kHz inducing a latency requirement that cannot exceed 0.2 to 1ms. Then, the stability control loop requires a control frequency in the 100 to 10000Hz range, implicating the need for a latency below 1 to 10ms. Other control loops exhibit frequencies that are below 30Hz and consequently implying affordable latencies starting from 33ms. Grounded on the requirements, the authors propose to offload some of these control loops according to latency expectations.

To conclude, this overview of the literature in the cloud-robotics area reveals some first insights about the low-latency expectations that currently induce barriers in the possible deployed solutions. Enabling a latency reduction in the communications between robots and central control would presumably open new opportunities in this field by allowing a new share of responsibilities between robots. This could result in systems that can be made more simple and low-cost, and control that can allow offloading more processing.

2.9 Synthetic characteristics of low-latency applications

Table 1 synthesizes the characteristics of low-latency applications based on the state of the art studied in the previous sections. In this synthetic table, low-latency applications are organized in column according to the classification presented in Figure 1 while its lines report the traffic profile characteristics of these applications such as their bitrates, transport protocols, acceptable latency ranges and packet-loss rates and whether the applications are adaptive.

Having a characterization of the traffic profiles of low-latency applications allow the project to analyze what are the latency issues faced by low-latency applications in given network conditions.

			Interaction between machines	
	Between humans	Interactions of humans with systems		
			Telerobotic	
Typical uplink bitrate	[0.1 - 2] Mbps per video stream; [5 - 100] Kbps per audio stream[10]	[50 - 150] Kbps[11]	[1 - 2] Mbps[12] for video-based control + [0 - 50] Mbps[13] for payload streams	[60 - 100] Kbps for the C&C stream
Typical downlink bitrate		[3 - 40] Mbps[17]	[60 - 100] Kbps for the C&C stream	[1 - 2] Mbps[12] for video-based control + [0 - 50] Mbps[13] for payload streams
Typical transport protocol	RTP/UDP for audio and video	RTP/UDP for audio, video and game control	RTP/UDP for audio, video and C&C	
End-to-end latency constraint	Human interactions	Player satisfaction and performances	Drone speed, acceleration capacity and sensor range[18]	
Typical end-to-end latency ranges	[150 - 400] ms for mouth-to-ear and pixel-to-eye	[60 - 200] ms	[0 - accidentogene] ms[18]	
Adaptive uplink bitrate ?	Yes for video; No for audio	No	Not found (likely No)	
Adaptive downlink bitrate ?		Yes for video; No for audio		
Acceptable packet loss rates	[0 - 10]% [19]	[0 - 10]% [19]	Not found	

	Interactions of humans with systems		Non-interactive
	Telerobotic		
	§2.7-Medical telerobotics: robot side	§2.7-Medical telerobotics: operator side	§2.3-live video streaming
Typical uplink bitrate	[1 - 100] Mbps[14] for A/V streams + ∼368 Kbps[15] for a 3DoF haptic stream	∼368 Kbps[14] for a 3DoF haptic stream	A few Kbps[16]
Typical downlink bitrate	∼368 Kbps for a 3DoF haptic stream	[1 - 100] Mbps[14] for A/V streams + ∼368 Kbps[15] for a 3DoF haptic stream	[1 - 30] Mbps for flat video; [10-300] Mbps for 360 video
Typical transport protocol	RTP/UDP for audio, video and haptic media		DASH (HTTP)
End-to-end latency constraint	Stability of closed-loop haptic control		Age of information
Typical end-to-end latency ranges	Not found		[60 - 200] ms
Adaptive uplink bitrate ?	Not found (likely No)		No
Adaptive downlink bitrate ?			Yes
Acceptable packet loss rates	0%[20]		Translates into delay

Table 1: Synthetic characteristics of low-latency applications

[10]When using Selective Forwarding Unit based conferencing, the endpoints can receive multiple audio-video streams

[11]Game dependent

[12]Derived from communication services and amputated from lower bitrates, assumed unpractical for drone piloting. Automated piloting of dumb drones (i.e. drones unable to take short-term decisions) is assumed to be similar to human drone piloting, except that video sensors also includes stereo frame based cameras or event-based cameras in which case the required video bitrate could be higher than the one referenced here

[13]According to 3GPP

[14]Depending on the number and quality of media feeds

[15]16 bits per DoF at a 1kHz sampling rate, i.e. 48 Kbps for 3 DoF at payload level and 368 Kbps at IP level

[16]HTTP GET requests

[17]Game, platform and network capacity dependent

[18]For instance, 250ms of button-to-pixel latency allows humans controlling drones at speeds up to 7 m/s (25km/h)

[19]Codec dependent

[20]Packet-loss can jeopardize the stability of closed-loop haptic control

3 Clustering of applications

The state-of-the-art of low-latency applications enables us to highlight the diversity of the applications in terms of constraints and requirements. In this section, we cluster these applications according to the causes of latency we will investigate in the project in order to improve the delivery of the contents. We first describe the main causes of latency before focusing on the network ones, taken into consideration in MOSAICO, for grouping the applications into well-identified clusters.

3.1 Main causes of latency

When speaking about low-latency(LL) applications, one should first identify where, when and how latency can increase or decrease in the end-to-end service delivery. This helps focusing on the more impacting parts when trying to improve the processing and the forwarding of the packets so as to ensure the required low latency.

[36] provides a taxonomy of the sources of delay for Internet-based applications, where five main categories are identified. This classification is approved by most of the research community and consequently it is the one considered in the MOSAICO project. In brief, the latency of the end-to-end service delivery can increase because of: 1) transmission links capacities (5G, 6G, xDSL, fiber, etc.); 2) the service delivery architecture (far remote server, use of caches, centralized/distributed delivery architecture, etc.); 3) the session between endpoints (TCP, UDP, fast connect, streams multiplexing, burstiness of the traffic, jitter, packet size, etc.); 4) the network equipment processing (routing, switching, queuing, scheduling, etc.); and 5) the endpoint processing (adaptability of the service to variations in the network conditions, video encoding, head-of-line blocking, number of parallel sessions, etc.). We describe hereafter these 5 categories.

Structural delay: Internet is a communication system that implies several network functions and many nodes and endpoints. Chained Network Function placement, content placement, routing, name resolution and path occupation rate is likely to produce some sub-optimal performances and may injure E2E latency. Some of these sources of delay can be managed with an appropriate orchestration and the help of Network Function Virtualization (NFV) for network function placement, or with the help of Content Delivery Networks (CDN) for content placement, which is in relation with the cacheable feature of low-latency use-cases previously described.

Interaction between endpoints: Structural delay is mainly related to topology and placement. It can be static, or dynamic thanks to network programmability. Delays due to interactions between endpoints, however, are due to the communications: initialization of transport protocol, negotiation of options, end-to-end control, session opening, security session initialization and negotiation, metadata transfer and interpretation, message aggregation and so on. Depending on options enabled or the protocol versions of each endpoint that take part in an exchange, negotiation and initialization steps may take a while before the beginning of the actual data transfer. This is not an issue when starting a low-latency service, but can have dramatic impact due to connection resets or session re-initializations during service operation. An end-to-end protocol may need to trigger control interactions during a session to guarantee reliability requirements (packet loss recovery or information gathering about the characteristics of a path). Some solutions have been developed, such as Explicit Congestion Notification (ECN), in order to limit the possibility of retransmission and packet loss.

Working on the reduction of this category of delay means working on the low-latency features previously described, i.e.: reliability, packet size, number of parallel sessions and the adaptability of the service to network conditions variations. The reduction of this category of delay must be handled by the endpoints to work by choosing an appropriate congestion control implementation, and designing latency-aware services that can adapt themselves to variations in the network conditions.

Delays along transmission paths: This category of delay is linked to data transmission along the chosen path. It concerns every network equipment between endpoints (routers, switches and middleboxes) as well as links between them (signal propagation delay that depends on the table core material and the link capacity). Medium Access Control (MAC) can be used to control access to a shared medium, which is a technique mainly used in mobile networking where radio resources are shared. Serialization delay is also an important point to consider. Serialization and de-serialization involve the process of exchanging data between a network card buffer and the wire. The performance of this process may vary with network conditions. A poor rate adaptation of a wireless link can significantly impact latency. We can also mention switching and forwarding delays, that depend on equipment performance regarding the number of rules processed and the access speed of entries in the forwarding table.

Finally, the main contribution to latency in this category of delay is the queuing delay. This can be handled with a combination of several technologies. An appropriate congestion control implementation is required to adapt data emission to network capacity and quickly react in the case of increasing queuing delay, before the actual congestion occurs. To this end, Active Queue Management is needed to better control queue size and signal queue length evolution to endpoints. ECN is also needed to better express and describe the event.

Delays related to link capacities: End-users can access any service, wherever they are, whatever their device is, according to the now widely adopted ATAWAD (Anytime, Anywhere, Any Device) concept. However, this might not always be possible for all applications, especially the LL applications. For instance, one end-user at home, connected to a high bandwidth access network such as fiber, can enjoy such a service, but another user, far from the network's central office, connected with a narrow bandwidth xDSL access network, might not appreciate it because of limited network capacities. Similarly, with a good 5G (or 6G) radio link, the latency requirements for such LL services can be ensured, but not for end-users connected with radio link of poor quality or to a 4G network or via a satellite network. The network link capacities have strong impact on the good delivery of LL services.

Regarding a device, if the user is connected to a high latency access network, he/she will not be able to use the service, unless the device has multiple interfaces and can manage multipath delivery for ensuring LL services delivery. This multipath option is needed to be able to parallelize the packet delivery and possibly reduce the latency. If this is a possible networking protocol solution, the device hardware needs to support it.

Another limiting factor for delivering LL services is the AS (Autonomous System) path, i.e. the different network actors between the end-users and the server. For instance, if the peering points between 2 AS's do not provide good connectivity, or the packet transits via an AS does not provide good enough network links, the latency will increase.

Intra-end-host delays: In this category, we include all the delays which are induced by the clients or by the servers. If, for some very simple applications they can be very low, for others it can reach a non-negligible time [49]. This also should be taken into consideration when deploying an LL application.

These delays include the application processing time, which is necessary for the service to behave as expected. But some services can do additional processing, not for the application execution itself, but, for instance, to adapt the delivery to the network conditions. Typically, we can mention the services which can encode their content differently depending on the available network bandwidth, the estimated end-to-end delay, etc. Video streaming applications are examples of such applications.

Another induced delay can be due to the session management. For example, a Web server implemented in HTTP/1.1 will suffer from the head-of-line (HOL) blocking, which should be avoided using HTTP2 or QUIC, with the parallel management of established sessions. This could also be the case for applications sequentially managing sessions with their end-users, where one that takes more time will penalize and add delay for all the others end-users.

Finally, the latency can be simply induced by the characteristics of the endpoints or servers. If a server is an old one, e.g. with a slower processor and fewer RAM, it will be slower than a more recent computer and thus induce delays in the end-to-end delivery.

3.2 Clustering of applications according to causes of latency tackled in MOSAICO

In the MOSAICO project, our objective is to build a solution to ensure the delivery of LL services. However, we will not be able to address all five categories which can impact the end-to-end latency. Indeed, it is not the goal of MOSAICO to investigate the link capacities (optical improvement, better radio scheduling, etc.) nor the server processing time, but we will focus on the networking delays. For this, we will investigate solutions for reducing latency imputable to structural delay and to the interaction between endpoints and along the transmission paths. More precisely, the solutions investigated in MOSAICO will be related to content placement (network proxies and caches) and service architecture (cloud server placement, cloud cache placement) for structural delays and to queuing delays for delays along transmission paths.

With the studied categories of low-latency applications, live video streaming (§ 2.3) is the only one for which content placement and cloud cache placement are a concern. However, we observe that for live video streaming, the dominant cause of end-to-end latency lies in the player's buffer. Therefore, we exclude this category from our clustering of applications according to causes of latency. The remaining causes of latency that are of concern for the clustering of low-latency applications are therefore: cloud server placement and Queuing delays.

A queuing delay appears when the bitrate of an application is above the capacity of the network. This leads us to cluster low-latency applications according to their bitrates and relatively to the network conditions (more precisely the network bottleneck capacities) in which they are intended to be used.

As both the network bottleneck capacity and the application bitrate can be time varying, this leads us to cluster applications according to how they adapt their bitrate to the network conditions. It turns out that the transport protocol is a sufficient discriminant. With RTP, the congestion control influences directly the application content bitrate (e.g. CODEC reconfiguration) while with TCP/QUIC, the congestion control doesn't have such a direct interaction (which may or may not be performed with another control loop at the application layer).

Finally, assuming that datacenters are already near the peering-points of the Internet Access Providers (IAP), cloud server placement within the IAP network can only be improved by a few milliseconds of latency. Therefore, in order to have a significant contribution within the end-to-end latency of an application, the application itself must have an extreme low-latency requirement. This leads us to cluster applications according to their latency requirements, for instance: low-latency applications where <100 ms of network RTT can bring QoE improvements; and, extreme low-latency applications where <10 ms of network RTT can bring QoE improvements.

Not to define solutions for just one application, but rather to a group of applications and since LL applications can be very wide, we cluster hereafter the main LL services based on their characteristics related to different latency classifications we will address, shown in Table 2. Our survey enables us to identify 3 main clusters, 2 being for low-latency applications, depending on their adaptativness and 1 cluster for extreme low-latency.

Clusters		Applications
Low-latency applications (<100ms network RTT can bring QoE improvements)	A- Queue building applications over TCP/QUIC	Realtime VR streaming over RTMP
	B- Queue building adaptive applications over RTP	cloud gaming
Extreme low-latency applications (<10ms network RTT can bring QoE improvements)	C- Haptic applications	Telerobotics with haptic control

Table 2: Clustering of low-latency applications

To improve the structural delay, we will deeply study on how to reduce the structural delay with the design of an orchestrator able to dynamically handle the network function placement in section 4.2. This will be associated with a work on microservices, aiming at a finer deployment of components. The technology choices for hosting the microservices will be P4 and OpenNetVM, introduced in section 6.

The delay resulting of interaction between endpoints and along transmission paths will also largely be addressed in the project, by facilitating the awareness and expression of the variations of network conditions. To this end, Active Queue Management (AQM) will be studied to better control queue size and signal queue length evolution to endpoints, and ECN will also be included to better express and describe the event, as introduced in section 4.1. Finally, MOSAICO will investigate the integration of novel techniques that permit zero-copy packet processing based on DPDK [74] in its framework, introduced in section 6.

4 Network solutions to reduce latency

As introduced in the last section, there are mainly two causes of latency, the time spent by the packets in queues and the latency between the client and the content server. In this Section, we present the current main solutions that aim to reduce this latency by: 1) reducing the queue delay via network algorithms, architecture or protocols; 2) improving the deployment and placement of network functions via NFV; and 3) specific environment solutions.

4.1 Reducing queueing delays

This section presents the main current solutions for managing the queueing delays in the networks. First the endpoints mechanisms to adapt the senders' bitrate to the network conditions (i.e. to the bottleneck capacity and to share bandwidth with concurrent traffic), known as Congestion Control Algorithms (CCA), are presented. The most recent solutions can now interact with network equipment and use explicit congestion signals. This is possible via Active Queue Management (AQM) and Explicit Congestion Notification (ECN) solutions, presented after.

4.1.1 Control congestion algorithms

In order to enable a connection to benefit from the best conditions without impacting other connections, TCP has four parts in it's Congestion Control and Error Recovery Algorithms, namely: slow start, congestion avoidance, fast retransmit, and fast recovery [26].

Typical TCP CCA such as Reno and Cubic are "capacity seeking", i.e. they probe the bottleneck capacity by increasing the bit-rate until a packet loss is detected following an AIMD (Additive increase/multiplicative decrease) mechanism during the congestion avoidance phase. It linearly increases the congestion window until a packet loss is detected, i.e. additionally increasing the MSS (maximum segment size). It strongly decreases its bit-rate when congestion (packet loss) is detected by a certain multiplicative factor (<1).

CCA implementations are required to detect network condition changes. There has been many CCA implementations [160] since the first one, each one improving the previous ones by being more reactive, more precise, etc. Subsequently, we briefly mention the main CCA implementations.

Reno [10] is one of the first congestion control algorithm. It was widely deployed on the Internet in 1990. It works on the AIMD principle and detects congestion only based on the loss of packets in the network. RENO's performance is correct when the loss rate in the network is low, but when the number of losses increases, RENO's efficiency in keeping the best throughput decreases sharply due to successive decreases in the window sizes. The main reason is that the RENO algorithm was designed to detect a single loss during the transmission of packets in one congestion window. When there are several packet losses in the same transmission window, RENO considers that these are several distinct congestion events and makes several decreases in the size of the congestion window. However, these multiple losses are often due to the same congestion event. As a result the reduction in the bitrate is too strong and the performance when using RENO is degraded. To improve the correction of multiple losses, the SACK (Selective Acknowledgments) option was proposed that allows sending the acknowledgments of all the packets received even if these packets have discontinuous sequence numbers which lets the sender know what packets are missing and retransmit them.

To increase the performance of RENO in case of many losses, newRENO [71] has been proposed. The improvement of NewRENO concerns the action to be done when receiving three "DUP ACK" packets. More specifically, when the algorithm enters the fast recovery phase, it remains in this state as long as all the packets in the congestion window are not acknowledged. Only once they are acknowledged, it decrease the size of the congestion window. If several losses are committed in the same congestion window, then newRENO considers them as a single loss and realizes a single decrease in the size of the congestion window.

Vegas [32][10], is faster since before loss detection, it evaluates the delay between the sender and the receiver to estimate queue status and eventually reduce its bitrate if the delay increases.

With the evolution of the Internet, the different variants of RENO and NewRENO showed limitations for high speed networks. BIC (Binary Increase Congestion) [190] solved the problem with its binary search algorithm. After a packet loss, BIC will reduce the window size by multiplying it by a factor $\beta < 1$. The window size just before reduction is set as the maximum size of the window named W_{max} and the window size after reduction becomes the minimum size named W_{min}. Then, BIC will perform a binary search between its two parameters (W_{max} and W_{min}) by making jumps to the value which is just half of W_{max} and W_{min}. If no loss is detected this new value is the new minimum window size. This process continues until the increase in the window is smaller than a fixed constant number. BIC then enters a phase called "max probing" where the window size slowly increases for some time before changing to a linear increase if no lost packets are reported.

Unlike RENO and newRENO, BIC no longer relies on receiving acknowledgments to increase the size of the congestion window, so the RTT no longer impacts the flow increase phase. BIC's performance is correct in networks with high throughput and latency, but it is too aggressive for small throughput and short RTT networks.

CUBIC [82] is an evolution of BIC and modifies the increase in window size by a cubic function (hence its name). After a packet loss, CUBIC will reduce the window size by multiplying it by a constant β factor with $\beta < 1$. Then, CUBIC will increase the size of the congestion window by using the concave part of its cubic function. In this way, the plateau value (W_{max}) is reached and the algorithm stabilizes the flow around this value which is the transition between the concave and the convex part. Finally, CUBIC will increase the size of the congestion window using the convex part.
CUBIC is less aggressive than BIC, which makes it more "friendly" to other versions of TCP. CUBIC is also simpler because it has only one algorithm for its rising phase, and does not use acknowledgments to increase the window size. This is what makes CUBIC more efficient in low speed networks or with a short RTTs.

BBR (Bottleneck Bandwidth and Round-trip propagation time) [45], promoted by Google, changes the principle of the congestion control algorithm and does not consider losses as a sign of congestion. To ensure that the link is not congested, it adapts its throughput based on a model using bandwidth and delay estimation following two phases. A mechanism is used to assess whether an increase in throughput causes more congestion on the network. To do this, it modifies the congestion window during an RTT to 1.25 times the bitrate and over the following period to 0.75 times the bitrate. By observing the evolution of the current RTT, BBR can determine if the available bandwidth of the saturation point has increased or if the throughput remains unchanged. BRR is currently adopted by many actors and is used with the QUIC protocol (i.e. Google services).

4.1.2 Active queue management

The previously described CCA implementations work independently of the network equipment. However, the network providers also implemented improvements in their switches, routers, and typically added the so-called AQM (Active Queue Management) mechanism.

Historically, the network equipment queue management was simple: it consisted in just applying a drop-tail model (i.e. a packet is added to the queue only if space is available, otherwise it is dropped). With AQM, the queue management is smarter and used the status and the evolution of the queue size. In this way, the network equipment can take actions (drop or mark the packet) before the queue is completely full. This is useful since while end-to-end latency has multiple causes, the reduction of queuing delays is reckoned to offer one of the best performance-gain-to-deployment-cost ratios [36]. The queue analysis can be based on the queue size (Bytes) or time (ms), depending on the AQM algorithm.

After several years of existence, the IETF has published general recommendations [16] to improve the performance and security, specifically for flows that are unresponsive to congestion notifications for dropping or marking packets. Amongst the different AQMs that exist [3], we can mention the following ones.

RED (Random Early Detection) [72] is the first AQM algorithm developed in 1993 and implemented in the Internet. Its primary goal is to avoid network congestion due to bottleneck links, that is, throughput limitation due to the capacity of the link. It acts on the queue in the equipment, when the packet arrives. It looks at the size of the queue (in number of packets or in bytes, depending on the implementation) and decides to mark the packet, based on the compute probability if the average queue size is between a min and a max value, or always marking it if the queue size is more than the max value. Marking the packet will let the destination host know that the network is congested. It will therefore report it to the sender, which will adapt its bitrate. The equipment can also make the decision to drop the packets if the flows last less than 1 RTT or if the sender is unable to regulate its bitrate (e.g. if the return packet is lost).

BLUE [68] is an AQM dating from 2002. It was designed to overcome the shortcomings of RED. Like RED, BLUE relies on the length of the queue, in terms of packets or bytes. The goal of BLUE is to: (1) minimize packet loss and delays; (2) avoid global source synchronization; and, (3) maintain high link utilization. These are the same goals as RED, but tests conducted on both shew that BLUE converges to the optimal point faster than RED. Four parameters are important: the threshold used for starting to drop packets, the update time, delta$_1$, and delta$_2$. BLUE looks at the length of the queue when inserting (or removing) the packet. If this is greater than a certain value (threshold), and we are at $t + t_{shift}$, the algorithm increases the probability of delta$_1$; otherwise, it does nothing. This increases the probability of steadily dropping, giving the network time to regulate. Then, the packet is dropped with some probability. As long as this is not zero, all packets have a probability of being dropped. If the queue is empty, we subtract delta_2 from the probability P. This makes it possible not to reset it immediately, and to avoid a new congestion.

Codel[139][140] is an AQM algorithm introduced in 2012. Unlike RED, this algorithm is not based on queue size or queue occupancy, but operates on queue delays. We set a limit (target-time) to the time spent in the queue (sojourn time) by the packet. Once the packet is out of the queue, Codel measures the time spent in the queue by the packet. If this time is above this target value y, for a time T, then the program makes the decision to drop the packet and configures $T = T/count$, where *count* represents the number of times packets have been dropped since the network equipment started to drop packets (i.e. is in ”drop-state”). This allows a linear evolution at the beginning that then becomes exponential, thus making it possible not to immediately drop all the packets and give the network time to regulate itself. Once the sojourn time of the packet goes below the threshold, it leaves the ”drop-state”. FQ-Codel [84] works like Codel. The only difference is that each ”flow” (a flow being defined by the ”5-tuple” srcAddr, dstAddr, srcPort, dstPort, protocol) is stored in a queue of its own. Then, a Codel-type algorithm is applied to each queue.

The Proportional Integral (PI) controller [85] was designed in 2001, inspired by linear system analysis. It aims at maintaining a target value for queuing delay in a responsive and stable manner.

PIE [144] [143] is an evolution of PI developed for low-latency architectures. The idea behind this algorithm is to react more quickly to congestion and prevent it. It aims to keep the delay in the queue below a given threshold. If this value is exceeded, the packet is dropped with a certain probability. After a given time (usually 1 RTT), a new computation checks this value. If it is still too long, the probability of ”dropping” the packet is increased by a delta value. The PIE controller computes a delta p value from a term accounting for the queuing delay variation since the last time and a term accounting for the difference between the current queuing delay and the target delay. These terms are respectively multiplied by the variables α and β and then added. At each time instant T, this delta p value is added to the previous p value to obtain the new drop/mark probability. At each turn, the α and β are adjusted by the algorithm. If they are not, the probability of dropping increases too quickly and causes large variations.

PI2 [59], defined in 2016, is based on the PIE principle. PI2 allows one to have two fixed values for α and β, by squaring the probability to adapt it to the sensitivity of the signal (i.e. the error between the target delay and queue time). This makes it possible to have higher α and β values without creating a disturbance in the probability and therefore making it possible to react more quickly. There are two ways to model this squared rise in this algorithm: either by raising the probability squared (this can be slow, depending on hardware constraints) or by using two random variables. If the probability of dropping is greater than the maximum of these two variables, then the packet is dropped/marked. The idea is to ”think twice before dropping a packet”. If the tail delay increases, the probability also increases. This allows reaching a probability of 1 as soon as the target time is exceeded.

4.1.3 Explicit congestion notification

Many congestion control algorithms (Reno, Vegas, Cubic, BBR…) rely on implicit congestion signals (i.e. packet loss and delay).

ECN (Explicit Congestion Notification) [73] is an alternative where network equipment can explicitly indicate possible congestion to endpoints. ECN is managed by 2 bits in the Traffic Class field of the IP header, and endpoints should support ECN to adapt their throughput (bit specified and announced during TCP negotiation). When an AQM-enabled network device detects congestion, it can provide an explicit signal, at the congestion onset, by marking packets with a flag in their IP headers (ECN bits put to 11, meaning Congestion Encountered). The receiver of a marked packet echoes ECN information (ECE bit) to sender (through a field in the TCP Header in all ACK packets) until the sender has reduced its bitrate and announced it through the CWR bit.

However, classic ECN which sends binary signals per round-trip-time only allows slow and coarse reaction. This is because the receiver can only report to the server that there is congestion in the network. It cannot analyze the severity of this congestion. To improve this, IETF defined Accurate ECN [34]. In this case, the network device still marks packets with the flag in the header, but the receiver reports to the sender the proportion of marked packets (rather than a binary signal). Thus, the sender can more finely and faster react according to the congestion level. For this, Accurate ECN adds additional information on the congestion, such as the number of received CE packets with payload in ECN TCP bits (ACE) and the number of received ECT(0), ECT(1) and ECE packets in a new AccECN TCP Option. This allows the sender to know the number of marked packets and to decrease more finely its throughput, making it possible to only reduce by a fraction (depending of estimated congestion) instead of half of the bitrate as done by classic TCP.

This new Accurate ECN however requires a new TCP stack (such as DCTCP) to manage the new ECN behavior. In [35], the authors detail how to adapt DCTCP [20], an accurate ECN-based congestion control designed for data centers, to make it safe for deploying it over the Internet. The result is known as TCP-Prague [35], a new TCP congestion algorithm that is still in the progress of being implemented to become part of the scalable congestion control algorithms. The main advantage of this new TCP CCA is its reactivity and fine adaptation to congestion but its main drawback is that it can monopolize all the available bandwidth and impact other traffic.

4.1.4 Low latency low loss scalable throughput

To improve this coexistence of different traffic types, L4S (Low Latency Low Loss Scalable throughput) [38] has been designed. The L4S architecture makes Accurate ECN traffic possible and allows the sharing of bandwidth with other traffic types, while still ensuring low-latency delivery for ECN scalable packets. IETF has defined an L4S implementation called ”DualQ Coupled AQM” (see Figure 17).

The first part of the L4S architecture (IP-ECN) serves to classify incoming packets according to their priority (low-latency or Best-Effort flow). This is done according to the ECN bits of the diffserv field of the IP header. The packets thus go to two different queues.

In this way, two different AQMs can be introduced. The IETF does not impose to use one particular AQM algorithm, but in their implementation they propose using PI2 [59]. low-latency packets can be marked, with some probability, but are never dropped, while Best-effort packets can be marked or dropped. The probability is different for the two types of flows. For low-latency flows, the probability used is the maximum between the probability output by the ”native AQM” (LL traffic) and the probability of the ”base AQM” (BE traffic) multiplied by a factor K (by default, $K = 2$). This enables taking into account what happens on the classic side, so as not to penalize it excessively. This is where the word ”Coupled” takes on its full meaning. It does not just act on low-latency traffic, but also considers what is happening with other traffic types. The decision to mark/drop conventional packets is made based on the ECN bits. These bits, included in the ”diff-serv” field of the IP header, are used to indicate whether there is congestion or not. If the packet is marked, it means that the equipment is congested. The information will therefore be transmitted to the destination host.

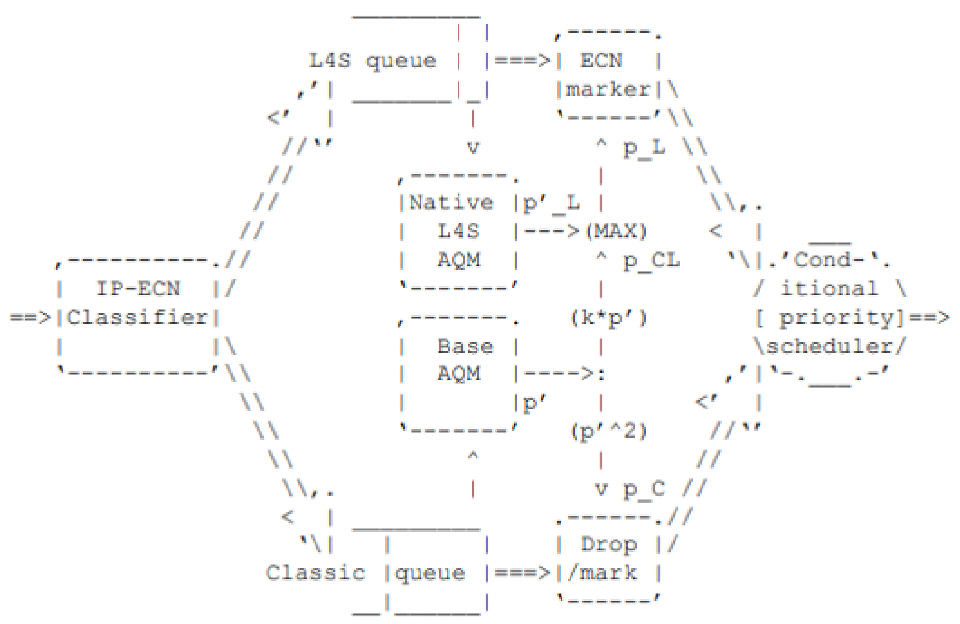


Figure 17: L4S architecture: dual queue coupled AQM [38]

4.2 Network functions deployment and placement

As a relevant means to address latency issues, the deployment of network functions at the ”best” location, given the related services requirements, has to be considered. Such an issue has largely been studied in the context of Network Function Virtualization (NFV - see Section 4.2), with specific studies on placement, routing and scheduling that we review subsequently.

4.2.1 Different issues related to the deployment and placement of network functions

The deployment and placement of network functions mainly allows the placement of virtual network functions in different nodes, the routing of flows through these functions, and their scheduling. For this purpose, it can be divided into four mains issues. In the following, we propose a classification of the orchestration services based on VNF-P (placement), VNF-C (chaining), VNF-S (scheduling) and VNF-PR (placement and routing) that we first define as follows:

VNF Scheduling (VNF-S): Based on Brucker [39], scheduling consists of finding time slots on which activities will be treated within certain given constraints. Among these constraints we can find priority constraints or resource limitations. With respect to the network function orchestration context, the objective is to find time slots for the execution of each network function that makes up the SFC service requests. The execution is done on a set of hardware resources (physical or virtual machines). Figure 18a shows, as an example, three SFC requests, each composed of three or four functions, and each function has a time slot allocated on the nodes that allows respecting their execution order.

VNF Chaining (VNF-C): The problem of virtual network function chaining (VNF-C) deals with the different mechanisms of chaining and steering of the corresponding flows in order to reach their destinations [41]. In a simpler way, the VNF-C issue allows creating a path between the different nodes of the network for the processing of each request in an optimized way, i.e. minimizing the cost associated to the use of a node or link. Figure 18b depicts an example with an SFC query consisting of two VNFs. We can see that according to the instantiation of VNFs on nodes, there can be several paths through which the requests can go through. As such, the chaining problem consists in determining the ”best” path for requests.

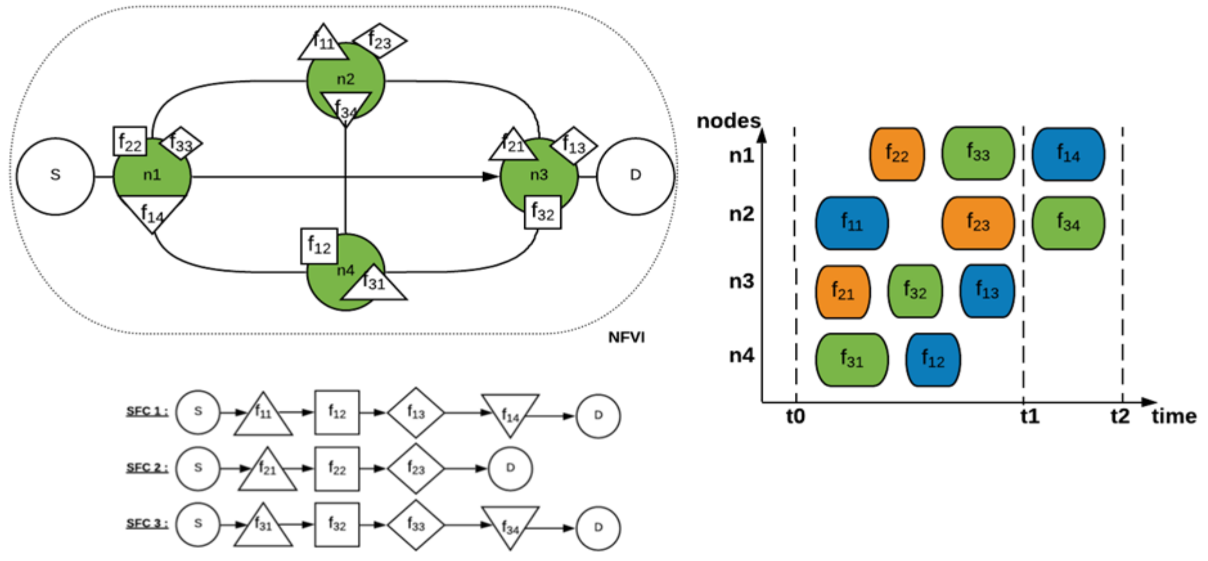
VNF Placement (VNF-P): In NFV, a critical problem is the VNF placement issue (VNF-P) which consists in determining the optimal placement of VNFs on the physical servers in the network, most of the time under the constraint of processing requests for services with a required quality. This VNF-P issue is an *NP*-Hard problem [17]. In Figure 18c, one can see an infrastructure containing three N-PoP that may receive VNFs. We also have two VNF instances to place on the node and consequently, the problem is determining the number of instances needed and the nodes where to place them.

VNF Placement and Routing (VNF-PR): The VNF-PR issue involves two types of decisions which are (1) the placement of VNF instances on the network nodes and (2) the routing of flows through the different VNF instances. In the case where an order has to be respected for each SFC, the problem implies considering the VNF-P and VNF-C issues at the same time.

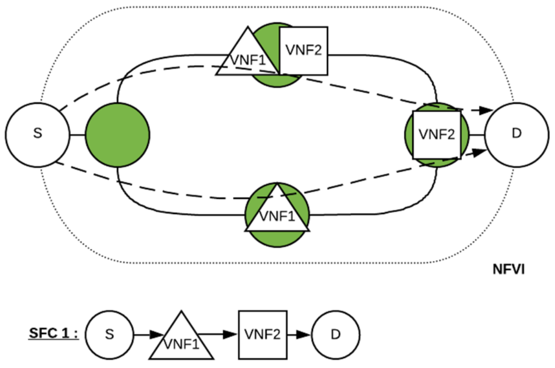
Given these definitions of orchestration issues, their classification is depicted in Figure 19, where we start by separating orchestration with and without precedence constraints. The latter concerns the respect of a certain order between the VNFs within the same SFC. We note that the VNF-S issue is not considered as being without precedence constraints because it necessarily involves an order between the VNFs to allow their scheduling. Then, we have subdivided the VNF-C issue with precedence constraints based on the capacity of the functions, we can consider or not that a function has to respect a limit in the number of execution instances of VNFs. When considering the capacity on the functions, there are two possible approaches: dynamic and static.

We have also grouped different issues together. The first one is between the VNF-P and VNF-R, which results in the VNF-PR issue and corresponds to the placement and routing without precedence constraints. The latter can also be considered with a single type of VNF or several types of VNFs which corresponds in our classification respectively to 1-VNF-PR and n-VNF-PR.

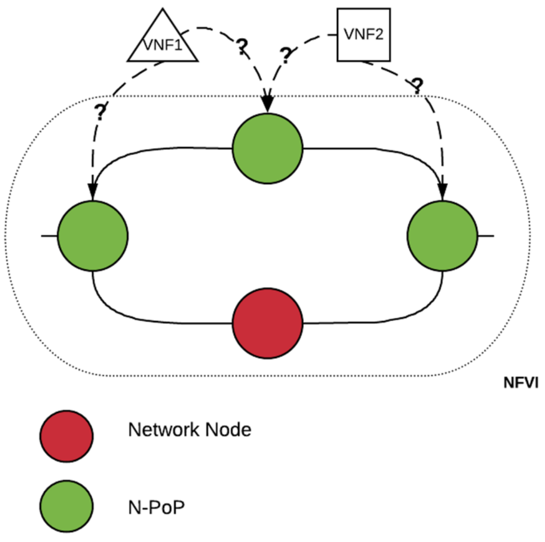
In parallel, we also grouped the VNF-P and VNF-C issues, which correspond to the placement and chaining. It should be noted that in the literature, the majority of the articles dealing with orchestration problems does not make the difference between the VNF-PR and VNF-PC problems. Finally, for the VNF-PC and VNF-PR issues, we have also subdivided them according to the capacity criterion on the functions.



(a) VNF-S



(b) VNF-C



(c) VNF-P

Figure 18: VNF issues examples

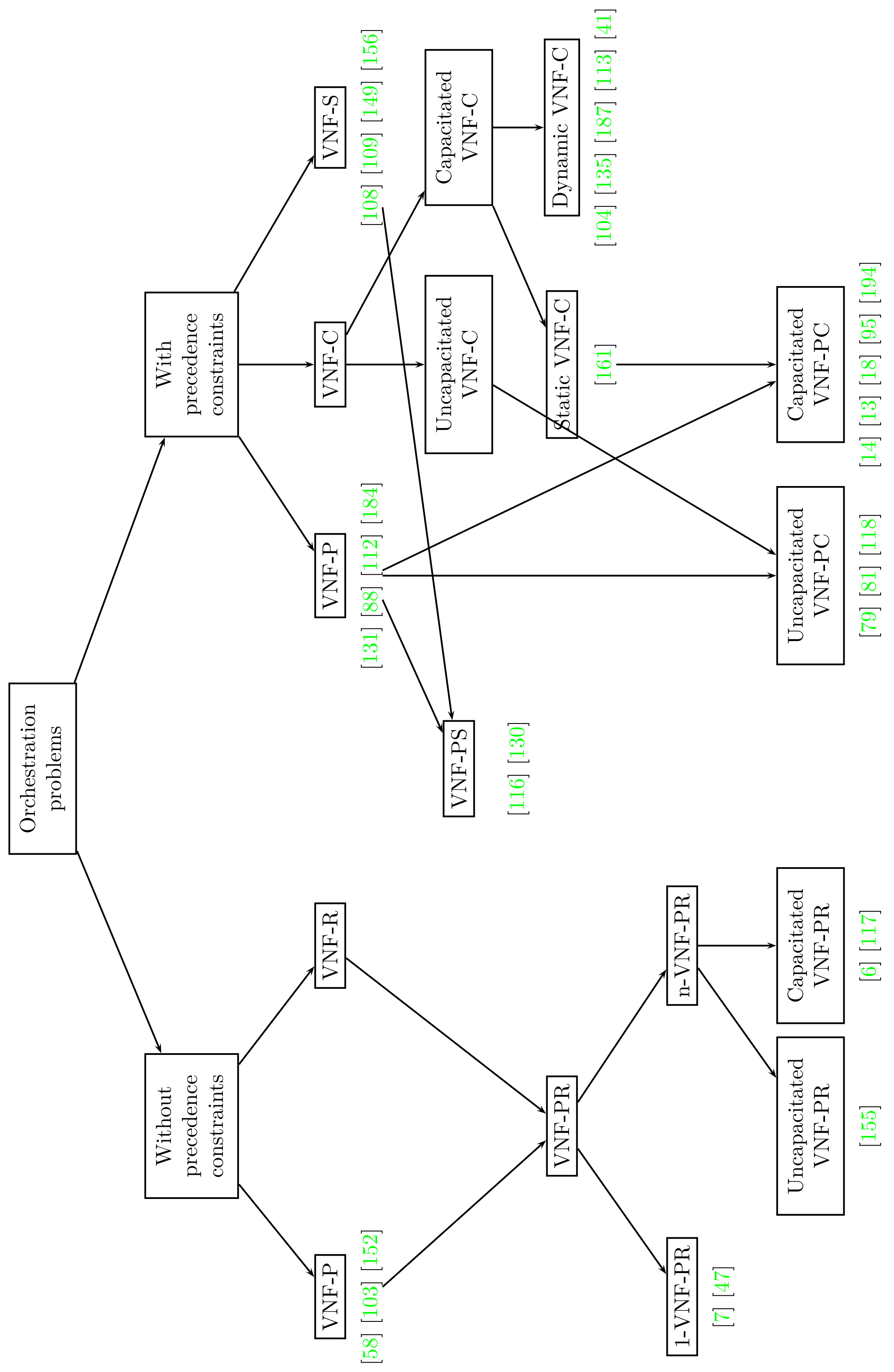


Figure 19: Orchestration issues classification

Besides, all of these issues can be addressed in different contexts whose most relevant features for the orchestration are:

- Type of network environment: The network may consist of physical and/or virtual functions. When both kinds of functions coexists, it is called a hybrid network;
- Type of architectural style: Network functions can be implemented as a monolithic fashion, but more recently, they are sometimes split down into low-coupled basic sub-functions that communicate with each other through APIs. In the latter case, we have what is called microservices.

Consequently, the deployment and placement of service functions raises interesting questions that can be addresses by optimization tools implementing, for instance, mathematical programming and heuristic algorithms. The main issue to solve is to maximize or minimize a mathematical function over a set of constraints. Accordingly, different articles studied the complexity of the different problems.

Complexity analysis The complexity study of the various problems related to orchestration exposed above represents a work of some importance because it allows to direct the research, particularly by guiding it toward the types of solutions that can be developed. Thus, various scientific articles dealing with orchestration analyze the complexity of specific problems [118, 58, 6].

In [8], Addis *et al.* study the complexity of different types of placement and routing problems. Particularly, they prove that UVNFP-SP (Uncapacitated VNF Placement with Simple Path routing) for the placement and routing of only one type of service (VNF) on simple path, without capacity on node and arc, is solvable in a polynomial time. The versions of the problem with capacity constraints on the nodes or arcs are for their part *NP-Complete*. The authors also discuss the impact of network topology (ring, hierarchical, etc.) on the complexity of VNF placement and routing issues. For example, on a ring topology, the VNF-PR problem without capacity constraints on the links and nodes has a polynomial solution.

Concerning the different problems of orchestrations (i.e. VNF-P, VNF-C, VNF-S and VNF-PR) with capacity constraints on the nodes and links, they all have been proven to be *NP-Complete* which means that there is no know polynomial algorithm that solves them. Consequently, the majority of the solutions in the field are approximate solutions because exact solutions would require exponential execution time and are therefore not scalable in real situations. In cases where exact solutions are presented, they are executed on small instances to validate some models or evaluate the quality of approximate algorithms.

Generic solving approach Orchestration problems can be modeled as optimization problems that aim at finding a feasible solution optimizing some objectives, such as the maximization of the quality of service, the minimization of the cost of infrastructure, etc. Thus, possible ways to solve them are:

Modelling through a mathematical program and using an exact method: This allows obtaining optimal solutions but they are often inoperative on large-scale instances because their execution time is relatively long (*NP*-Hard problems). Modeling an orchestration problem through a mathematical program leads to an ILP (Integer Linear Problem) or a MILP (Mixed Integer Linear Problem). A MILP is equivalent to an ILP except that it contains one or more variables that are not integers. The objective function tries to minimize or maximize a value under constraints that will refer to the different aspects of the problem and use some integer decision variables. Among the exact resolution methods for such models, we find algorithms like Branch-and-Bound or Branch-and-Cost, that can be called by using a solver (e.g. CPLEX, LINGO and GLPK [127]).

To understand what a mathematical program is, following is a basic example of an ILP model:

Objective function: Maximize y
Constraints:
$-x + y \leq 1$
$3x + 2y \leq 12$
$2x + 3y \leq 12$
$x, y \geq 0$
$x, y \in Z$

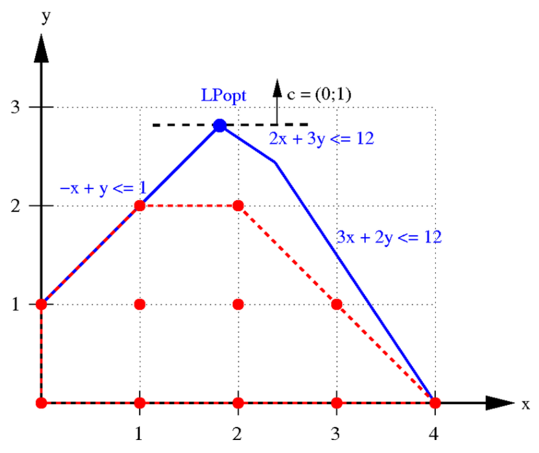


Figure 20: ILP example

Figure 20 illustrates the feasible domain (limited in blue by the constraints and in red by the nature of the variables, in this case positive integers). The black arrow is the gradient of the objective function. It indicates the direction of the optimal solution. The blue point (1,9; 2,9) is the optimal solution of the relaxed model where the variables are non-integer and consequently cannot satisfy the original problem. From this point, one may apply an exact algorithm such as the Branch-and-Bound one to build an exploration tree that explores the restricted feasible domain where x-variable must be smaller or equal to 1 on one branch, and explore the restricted feasible domain in which y-variable must be greater or equal to 2 on another branch, leading to the two red points (1; 2), (2; 2).

Developing approximate methods: This allows obtaining feasible solutions, hopefully close to an optimal one with an advantage that the execution time is relatively very short. For example Bari *et al.* [17] have tested two methods to orchestrate VNFs: an exact and an approximate one. The approximate method was between 65 and 3500 times faster for a solution close to optimal (within 1.3 times of the optimal solution). Among this class of methods, we find mainly heuristic or meta-heuristic algorithms. A heuristic algorithm is a constructive method based on problem-dependent techniques. It generally builds a solution step-by-step based on a greedy criterion, for instance. Most of the time, it is followed by a local search that looks for an improved solution by exploring the solution space close to the current one. To this aim, the local search explores what is called a neighborhood made of solutions achievable by a simple ”move” that is a small modification of the structure of the current solution. On the other hand, a meta-heuristic method explores broader solutions with higher level strategies (a ”meta” principle). This allows them to explore more extensively the solution space in the aim of escaping from local optima and thus to get a hopefully better solution than heuristics. These approaches may include schemes with several neighborhood structures, build or destroy procedures, or combining components of several solutions.

Certainly, we can find for each problem different and specific resolution approaches that we will describe in the following.

4.2.2 VNF scheduling

As previously defined, the VNF Scheduling (VNF-S) issue consists of finding time slots for the execution of each network function composing the SFC service requests. The approaches to solve this problem are based on an ILP/MILP specific model or reduced to problems already existing in operational research such as the limited resource-based project scheduling problem (RCPSP), as in [156] where the authors formalise the problem of scheduling VNF as job-shop problem that is known as solvable through an RCPSP.

Solutions can also be based on Reinforcement Learning (RL). RL consists of an autonomous agent learning the actions to be taken using experience, in order to optimize a quantitative reward over time. For example, in [109] Li *et al.* use such a technique (RL Q-learning) to find the near-optimal sequencing of VNFs. Regarding the modeling, they highlight the end-to-end delay and try to reduce it. The authors begin by formulating the problem through a NP-hard MILP and, due to its NP-hardness, they propose modeling based on MDP (Markov Decision Process). They then use an RL algorithm (Q-learning based algorithm) to find an almost optimal sequencing. The tests carried out make it possible to prove the effectiveness of the proposal by comparing it with the benchmarking of heuristic solutions.

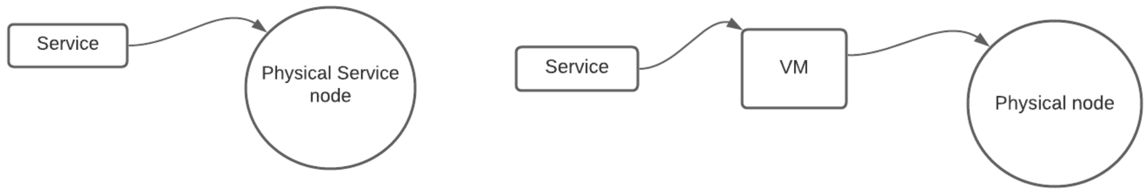
For their part, Chuan Pham *et al.* [149] propose a match-based algorithm in order to solve the problem of VNF scheduling and state that: *Matching algorithms are algorithms used to solve graph matching problems in graph theory. A matching problem arises when a set of edges must be drawn that do not share any vertices.* This approach ensures that all requests are met. The contribution of the approach is a 36% increase in the speed of execution of VNFs as compared to the conventional round valve method.

We also find articles, such as [116] and [130], that address the scheduling issue in conjunction with the placement issue. For example, Mijumbi *et al.* [130] address the scheduling problem and at the same time the placement. The authors propose three greedy algorithms to solve the problem: GFP (Greedy Fast Processing), GBA (Greedy Best Availability) and GLL (Greedy Least Loaded). The operation of these three algorithms is as follows: upon the arrival of a service request, the functions of the service are mapped and sequenced, i.e. the first is mapped and sequenced, then the next, etc. For each function i, all $N(i)$ nodes that have the ability to process it are determined. These nodes are then classified according to the processing cost criterion, i.e. least load for GLL, lowest processing times for GFP, and shortest virtual node queues for GBA. Then, the node with the best rank is chosen for mapping, and the function is scheduled for processing at the end of the node queue. However, the three proposed algorithms do not take into account the processing and transfer times on the links between the physical and virtual nodes. Moreover, the proposed solution is static, meaning that the scheduling and placement cannot change with the evolution of the network conditions over time. Additionally, there is a lack of optimisation which is essential in order to propose optimal solutions in addition to being feasible.

4.2.3 VNF placement

As previously defined, the VNF Placement (VNF-P) problem mainly consists in determining the placement of the VNFs on the physical network. The modeling is mainly done through an ILP or a MILP, but exact methods are not scalable for solving large instances. This is why, most of the times, the authors complete their study with an approximate approach.

For example, Bari *et al.* in [18] identify the placement and routing problem of VNFs that they refer to as VNF-OP (VNF Orchestration problem), and propose an ILP model for determining the number of VNFs and their locations in order to optimize the operational cost of the network and the use of resources. In a scenario of 23 nodes and 43 links, CPLEX required 1595.12s to solve the model while the heuristic-based algorithm spent 0.442s within 1.3 times of the optimal solution. Gouareb *et al.* [79] minimize the cost of delay on links. They consider that all VNFs associated with each SFC are located at the same node to avoid addressing priority constraints. In the same sense, Addis *et al.* [6] consider an objective function composed of two terms which are the minimization of the number of CPUs used, and the risk of network saturation, which induces the need for real decision variables leading to a MILP.



(a) Service deployment on a physical service node

(b) Service deployment on a virtualized service instance

Figure 21: Service allocation approaches

The literature also includes articles using only exact methods but on small instances. For example in [131], the authors deal with the VNF-P issue in an hybrid environment in which part of the services may consist of physical network functions and part of virtual network functions. Figure 21 shows the two possibilities of service deployment: directly on a physical component (21a), and on a virtual machine which itself is instantiated on a physical component (21b). The objective function aims at minimizing the number of servers used under 14 linear constraints. As the use of servers can be partial, the authors add a constraint that considers this and allows its minimization in the objective function. The evaluation uses CPLEX on a small instance (10 edge nodes, 5 switches and 4 core routers) with optimal solutions obtained in less than 16 seconds. However, the authors do not provide a resolution scheme applicable on large realistic instances.

Other articles propose a model but without solving it, preferring to directly develop heuristic approaches. For example, Riggio *et al.* [157] modelled the VNF-P issue through an ILP and they apply a heuristic-based approach to solve the problem, which is divided into three stages which are the choice of candidate nodes for placement, then their tries, and finally the placement. Also in [58], Cohen *et al.* address different variants of the VNF-P issue (with/without resource capacity on nodes/functions, etc.) by proposing various algorithms. In terms of performance measurement (i.e. the objective function), the authors rely on the cost of distance between customers and the virtual functions they are served by, as well as a second measure which is the cost of installing these functions. Note that the authors do not take into account the chaining constraints in order to optimize the placement as much as possible. In the same way, in order to further minimize the number of VNFs used, Sang *et al.* [162] also focus on the placement of VNFs without considering the constraints of chaining.

In addition to the use of heuristic algorithms, some proposals resolve the problem with approaches based on meta-heuristic algorithms such as Windhya Rankothge *et al.* [152]. Here, the authors propose a resource allocation algorithm for VNFs based on genetic algorithms. Compared to solution solving models (ILP or MILP) using a solver, the proposed algorithm is better in terms of execution time: a few hours for an ILP-based solution versus a few milliseconds for the solution proposed on an architecture consisting of 10 VNFs and 16 servers. However, it should be noted that the authors assume that all VNFs are implemented using Virtual Machines.

4.2.4 VNF chaining

As a reminder, the issue of Virtual Network Function Chaining (VNF-C) deals with the different mechanisms of VNF chaining and of managing the corresponding flows in order to reach their destination [78]
.

In the literature, there exists two approaches to deal with this problem: a static approach and a dynamic one that takes into account new queries and modifies the chaining done for queries already processed. Among the static approaches we find Sahhaf *et al.* [161] as well as Lee *et al.* [104] which formulate the problem through an ILP model with a multi-objective function. The latter aims to minimize the total cost and end-to-end latency. Also, Sahhaf *et al.* [161] propose a heuristic-based algorithm to solve the problem that can be considered in two parts which are: the selection of the decomposition consisting of prioritizing the decomposition of the VNFs, and the chaining of the VNF instances on the network nodes according to the defined strategy.

In dynamic resolution strategies, Liu *et al.* [113] address the issue by responding to new demands for services and readjusting ongoing demands. In order to achieve their goal, the authors build an ILP model and propose a heuristic method based on the generation of columns. The idea of the resolution approach is to break down the problem into a subset of problems in order to solve them iteratively. In [41], the authors also addresses the issue in a dynamic context. This results in the possibility for each service to add or remove VNFs. Modeling is done through an ILP. Concerning the resolution, in order to manage the dynamism of the context, two heuristic algorithms are proposed: the first allows to chain new requests, and the second allows to optimize the chaining already done, depending on the state of the network in real time.

We also find in the literature other articles that transform the VNF-C issue into another type of problem before solving it. Li *et al.* [110] handle the VNF-C issue as a grey theory problem. For their part, Wang *et al.* [187] transform the VNF-C problem into a Markov chain model and propose an algorithm based on an existing Markov approximation method in order to obtain solutions close to the optimal. Finally, Nam *et al.* [135] transform the VNF-C issue into an VNF clustering and attribution problem. This approach consists of grouping VNFs according to their popularity and then assigning service requests to clusters as needed.

However, in most cases the issues of chaining are dealt with the placement simultaneously to have an integrated problem called VNF-PR.

4.2.5 VNF placement and routing

The VNF Placement and Routing (VNF-PR) problem has two objectives which are: (1) the placement of VNF instances on the network nodes; and, (2) the routing of the different services in order to respond to all of their requests while respecting the various related constraints. In the majority of the related literature, the formulation of the VNF-PR issue is achieved through an Integral Linary Program (ILP) or a Mixed Integral Linaire Program (MILP). Concerning the resolution of the problem, some simply use a solver such as CPLEX [7, 81], or by developing an heuristic algorithm.

The term unicast defines a point-to-point network connection (i.e. from one host to an other). Several variants of the VNF placement and routing issue have been addressed in the literature for unicast communications. In [7], Addis *et al.* consider the placement and routing problem of VNFs with only one type of VNF for all applications. Therefore, all applications are equivalent. Nevertheless, they propose an extension of their work allowing to take into account different types of VNFs. In Casado *et al.* [47], the authors consider the problem, in a similar way as done by [7], with a single type of VNF and the use of an heuristic algorithm to solve it.

In [117], Luizelli *et al.* deal with the VNF-PR issue by using an ILP composed of 9 constraints reflecting the real constraints in the network. In this ILP, the consideration of latency minimization is translated to a constraint and is not formulated in the objective function. The latter is formulated with the aim of minimizing the number of VNF instances. The authors break down the problem into sub-problems and try to optimize the sub-problems. For this, they develop an algorithm known in the field of operational research as a VNS (Variable Neighborhood Search) that tries to find a near-optimal solution for each sub-problem.

Similarly, in [118], the authors keep the same modeling but they use a meta-heuristic-based algorithm that is more efficient. The evaluation done on an infrastructure with 200 N-PoP and a maximum of 60 SFC of 4 different types, an exact approach using CPLEX could not solve the problem in an acceptable time: for 18 SFC it takes more than 48 hours. By contrast, the heuristic-based solution could solve it in less than 30 minutes and considering the meta-heuristic solution, it takes less than 10 minutes to find a solution. Furthermore, the meta-heuristic solution is at most 2.1 times the optimal solution and the heuristic solution is at most 6.7 times the optimal one.

In [131], the authors minimize the number of activated nodes instead of VNF instances which shows the importance of the choice of cost measures in the formulation of the VNF-PR issue. The authors followed an approach that allows the problem to be solved in a reasonable time (30 minutes for 18 request and an infrastructure of 200 N-PoPs), with results very close to optimal ones. The heuristic first tries to find the optimal number of instances of VNFs, and then the algorithm follows an iterative approach such that in each iteration it seeks a workable solution and transforms the objective function into a constraint.

Other proposals deal with the problem of placing and routing virtual functions on certain network topologies. For example, Tomassilli [184] propose two approximation algorithms for a tree network topology. This allows to exploit the specific characteristics to obtain more efficient algorithms which cannot be used on all instances of the problem.

The number of constraints is a relevant aspect of modeling that allows the solution to be as close as possible to reality but induces more complexity for the resolution. Addis *et al.* formulate in [6] the problem of placement and routing as a MILP, taking into account a large number of constraints such as flow, latency, node capacity and function constraints. Regarding the objective function, it aims to minimize the number of links used to route the traffic and the number of nodes used to install the functions. For the solution, the authors propose an algorithm based on meta-heuristics. Zaid Allybokus*et al.* [14] also propose a solution that considers a generic version of the VNF placement and routing issue. As in [6], a large number of constraints (e.g. partial order of functions, end-to-end latency, conflicts between network functions, resource limitations and processing capabilities) are taken into account.

Concerning the objective function, one can notice that it changes from one article to another in order to adapt to the different research goals. For its part, Abhishek Gupta *et al.* [81] develop a model of the problem of placement and routing of virtual service function that minimizes the consumption of network resources. The guideline of the model, formulated as an ILP, is based on a network called NeC composed of NFV-capable Nodes and Data Center in contrast to [117] and [118] which do not take this aspect into account. This effectively reduces resource consumption.

In addition Sevil *et al.* [126] show that the problem of placement and routing of VNF could be considered as the localization-routing problem, which aims to create several paths between different VNFs, then connect them to get source-paths destination satisfying priority constraints. The localization-routing problem can be seen as a multi-product flow problem in which products can share some VNFs while minimizing the costs of edge node or path. The authors model the placement of the chained VNF problem as a mixed quadratic constraint program with three different objectives: to maximize the remaining data rate, to minimize the number of nodes activated, and minimize the latency of paths. The authors first consider the placement part of the problem. Then, to satisfy the web constraints, they create a path between the installed VNFs. The model was tested on small instances and solved using the

Gurobi Optimizer.

4.3 Addressing specific environments

We introduce in this section some specific solutions which can also help to reduce the latency for end-to-end communications.

4.3.1 Multicast communications

Multicast is an acknowledged approach for group communications which exhibits bandwidth economy. Nevertheless, several limitations exist which limit its wide-scale adoption. First, creating and modifying multicast trees requires more time and resources [111]. Especially, applied to an NFV environment, multicast communications require creating a multicast tree, placing the VNF instances on the network nodes, and directing the flow of requests through them. This is unlike the traditional multicast which consists in determining the path between a source node and different destination nodes [194]. Generally the construction of a multicast tree with a minimum cost is formulated through the Steiner tree problem which is an *NP*-Hard problem: *Given a set of P points, a tree for P is a network (i.e. a set of connected paths) such that all points are connected, and a tree is said to be Steiner if the total length of the network is minimal.* Nevertheless, different approximation algorithms are proposed. Vrontis *et al.* [186] build a Steiner tree using a shortest path. Byrka *et al.* [43] propose an approximation algorithm based on an ILP model and consider linear relaxation and rounding techniques.

Conventional treatment approaches based on ILP have been explored, as in [95]. As part of the VNF-PR issue, NarumiKiji *et al.* in [95] model the problem to be solved as an ILP for multicast service routing based on merging multiple service paths. They minimize the cost associated with the placement of VNF and the use of links in the provision of multicast service channels. They also develop a heuristic algorithm that can find a feasible solution within a reasonable time frame. We also find equivalent resolutions based on MILP modelling [13]. Omar *et al.* in [13] propose a MILP to model the problem of VNF placement and multicast traffic routing, while minimizing the deployment of VNFs and links under the physical constraints of network resources, flow conservation and VNF placement constraints. To reduce the computing complexity of the problem, heuristic algorithms are proposed, considering both the single path and multi-path routing.

For their part Zhang *et al.* in [194] translate the VNF-PR issue into a centralized problem through a global view provided by SDN technology. They propose three solutions: an heuristic algorithm, a dynamic heuristic method, and an exact branch-and-bound method. The first solution is based on an approximation algorithm and looks for the single NFV node that was used by all destination users. Then, it builds a minimum recovery tree among the end users. The second is a dynamic heuristic solution based on the algorithm of the shortest path. As for the third branch-and-bound algorithm solution, it aims at solving the ILP modeling.

4.3.2 Hybrid environments

With the deployment of virtual functions in networks, a multitude of researches have been carried out on the optimization of placement and routing of VNF in a full NFV environment. Nevertheless, the reality of network infrastructure is a mix of virtualized infrastructures and physical ones. As such, it is necessary to consider the capability of orchestration algorithms of dealing with hybrid environments composed of physical and virtual components to host service functions. In this context, a number of papers focus on hybrid scenarios [131, 88, 112, 103], proposing models that take into account both types of service functions and optimizes their simultaneous executions. The modeling methodology remains similar to full NFV modeling, unlike some constraints that allow to express the existence of a physical component. The model resolution is also equivalent to model resolution in full NFV environments. We can find solutions composed of heuristic-based algorithms [131] or algorithms based on Markov chains [88]. For example, Huang *et al.* in [88] study network service chaining in hybrid environments. The challenge addressed by the authors is to know how to efficiently chain the services in order to meet the demands and respect the constraints. For this, the authors design an algorithm based on the Markov approximation (MA). The tests performed prove that the algorithm can produce near-optimal solutions.

Similarly, in [112], Li *et al.* stress the difficulty of achieving a service function placement in a hybrid environment and implementing a common model. Here, the authors propose a MILP program and a heuristic algorithm to solve it. Larysa *et al.* in [103] deal with the virtualization of mobile networks and try to improve the efficiency of these through an optimal allocation of resources in a hybrid environment.

As in [131], which also focuses on the deployment of virtual network functions in a hybrid environment, the authors do not consider that the performance of the service instances depend on the amount of resources allocated. They explain how to optimally reconfigure the deployed network in a changing load environment.

4.3.3 Microservice environments

The microservice approach is an architectural style that structures an application into a set of loosely coupled services. Initially designed and deployed the cloud context in order to improve the flexibility and the ability to respond to the demand efficiently, it has been recently adapted to the NFV context to ease the reach of quality of service objectives.

In the initial Cloud context, there is a large number of papers dealing with the subject. Some of them deal with the problem of composing microservices in order to satisfy a demand in a cloud context. For instance, [198] deals with this problem in a Mobile Edge Computing environment by optimizing the end-to-end latency while reducing the resource consumption. The tests realized are based on latency and resources consumption costs. They show a better performance as compared to other algorithms of the same type. However, in comparison to algorithms that focus solely on optimizing latency or communication costs, the algorithm developed by the authors does not measure up to their performance. Other articles deal with the problem of decomposing services into microservices, such as [89], which proposes the decomposition of Content Delivery Network (CDN) components into microservices that communicate through RESTFul Web services and are provided as virtual network functions.

In addition to the decomposition of applications into microservices, the literature exhibits contributions dealing with orchestration issues in the cloud context. For example, in [22], the authors deal with the problem of scheduling microservices instantiated on VMs or PMs distributed over several Clouds. The paper takes into account latency, affinity constraint between microservices as well as the cost of resource consumption in order not to saturate the infrastructure and to disable the unused microservices. The algorithm developed is a heuristic that offers better performance than classical Greedy scheduling algorithms such as ”Least-full first with First Finish”. The paper [193] deals with the problem of routing and placement of microservices in the Cloud context with the objective of minimizing latency. The authors translate the problem into a Bin-Packing problem and propose a Greedy algorithm that has a lower performance as compared to other algorithms of the metaheuristic type which we could find in monolithic approaches. However, the developed solution remains complete from the orchestration perspective as it allows instantiation, routing and placement of microservices.

In the NFV context, we note in the literature that the microservice approach is of a gorwing interest and addressed form different perspectives. For example, we find some contributions that propose the decomposition of NFV orchestrators into a microservice architecture. In [176], the authors propose the decomposition of NFV Management and Network Orchestration (MANO) into microservices in order to achieve performance and availability criteria and above all to recover from failures. Indeed, with a microservice architecture, the occurrence of a failure on a component can be repaired quickly by precisely locating the faulty element and repair it or redeploy it. In [155], the authors also proposes an orchestrator architecture decomposed into microservices that allows automatic deployment and configuration of different service requests. It also allows the optimization of network resources. To that aim, it proposes two placement approaches supported by the orchestrator. The first is deterministic, based on an ILP, while the second is stochastic, based on artificial intelligence techniques. In contrast to [176], this orchestrator also has a component allowing the management and placement of VNFs based on microservices. However, it does not allow the optimization of the usage of VNFs decomposed into microservices, since it places all microservices related to the same VNF in the same location (N-PoP).

Concerning the context of orchestration of VNF based on microservice, there some very interesting and relevant first articles which pave the way for further contributions. The paper [83] is a pioneering paper in the field, the authors begin by presenting the challenges and the requirements of applying the microservice approach in the NFV context and then propose some possible solutions to meet these challenges. They also address the aspect of intra and inter connectivity of the servers in order to study the communication between the microservices that should be minimized to improve the end-to-end latency. Then, they propose an ILP placement model with the objective of minimizing end-to-end latency and to achieve this, they formulate the objective function through a minimization of the latency delay in the communication between the different microservices which they note as VNF Component (VNFC). They also propose, in their model, constraints specific to the end-to-end latency context as well as the NFV context. For example, they propose a constraint that will create an ”orbital” zone for each microservice before its placement. The microservice must be placed in this zone otherwise it will violate the latency constraints.

They also consider in the proposed model the affinity/anti-affinity constraints between microservices which represent the need to place certain microservices in the same node or on different nodes. Although no resolution algorithm is developed in this paper, it remains a reference on the subject by opening the way and directing future researches on the subject.

Following a similar purpose, the paper [52] defends the usage of the microservice approach in the deployment of VNFs and presents a state of the art on microservice architectures in the industrial and academic fields. The authors begin by proposing a decomposition of 5 VNFs (Wan Optimizer, Edge Firewall, Monitoring Function, Application Firewall and Load Balancer) into functionalities and deduct three important observations which are the overlapping of functionalities, the loss of CPU cycle as well as the inflexibility of the scaling. As for the state of the art, the paper presents four architectures. Two are developed in the academic field and are the Microboxes and OpenBox architectures and the two others are developed in the industrial field and are the Flurries and NetBricks architectures[3]. Then, the authors propose a comparison between the four architectures based on the following criteria: the placement solutions implemented, the fault tolerance, the communication mechanisms between the microservices, the memory isolation mechanisms used, and the procedures used in the transfer of ownership.

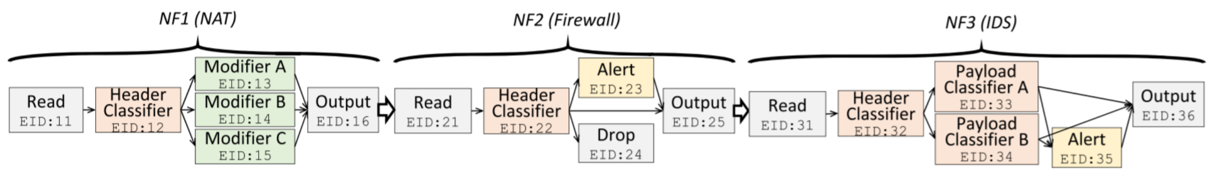
Still in the context of orchestration, in [172], the authors develop a placement solution that limits end-to-end latency and minimizes the exchange of messages between VNFs. It proposes a microservice aggregation based on the affinity between VNFs and microservices. Indeed, in aim to reduce latency, the paper focuses on minimizing the communication delay between the different VNFs or microservices and therefore creates network microservice bundles called Affinity Aggregates (AAs) which are a set of VNFs or microservices that have a strong communication between them and should therefore be deployed in a close way. For their implementation, the paper uses lightweight virtualization technologies such as containers. The envisaged approach seems promising except for the fact that it reduced the flexibility aspect of microservices deployment.

We also find some articles that already develop frameworks allowing the orchestration of microservices in the NFV context. For example [128] exploits the advantages of microservices (reusability, light weightiness, and better scaling) to overcome the performance degradation that NFV can generate. It proposes a MicroNF framework based on three axes. The first one consists of the reconstruction of SFCs upon reception in order to reuse microservices already present in the network and also to re-factor microservices when possible. For example, the SFC in Figure 22a comprises 3 VNFs: NAT, Firewall and IDS. The Header Classifier functionality is found within each of them. Consequently, a reconstruction of this SFC with microservice approach will allow to factorize this functionality and therefore reduce resource consumption and execution time as depicted in Figure 22b. For the second one, the microservices placement is the main issue the article tries to consolidate in the same node, with the largest possible number of microservices linked to a VNF. This consolidation aims to limit the communication between the microservices. Finally, for the third one, the paper proposes a new scaling approach that will attempt to balance the load between the VMs to avoid duplicating the microservice instances and thus limit communications. The paper also proposes an approach for scheduling the execution of different microservice instances. The proposed approach is incremental and changes from cycle to cycle to automatically achieve fairness and efficiency. Indeed, the algorithm will try to match the speed of packet processing with their arrival rates for each entity.

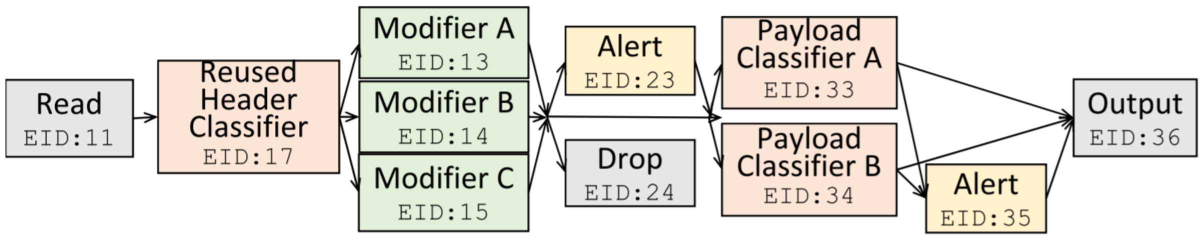
4.4 Overview of orchestration approaches

In order to conclude this part concerning the orchestration issues of network functions, we make a retrospective analysis on the different approaches adopted, the different constraints considered as well as the chosen contexts. The comparison between the different articles dealing with the same problem proved to be rather tedious because of the similarity of the resolution approaches. The main differences between the different papers generally lie in the constraints taken into account, the optimization objectives and the context in which they evolve. In order to inform this analysis, we provide in the following Table 3 a summary of the characteristics of the most important works studied in this state of the art. The most important characteristics chosen for classifying the different papers are :

[3]See section 6.2.1 for more information on these four architectures



(a) SFC with tree VNF



(b) SFC after rebuilding

Figure 22: Rebuilding SFC request with Microservice approach [128]

VNF-P : addresses the VNF-P issue
VNF-C : addresses the VNF-C issue
VNF-S : addresses the VNF-S issue
VNF-PR : addresses the VNF-PR issue
VNF-PS : addresses the VNF-P issue and VNF-S
Precedence : translates the order constraint between service functions for a service request
Function Capacity : translates the VNF limit processed by a function instance
Node Capacity : translates the function instance limit installed on a node
Arc Capacity: translates the limit of network links in terms of throughput
Conflict : reflects the fact that on some requests certain functions cannot be placed on the same node
Latency : reflects the fact that there is a maximum latency to be respected for each request
Exact Method : use of exact method for resolution
Heuristic Method : use of approximate methods for resolution
Minimizing Function Cost : minimizes the installation cost of function instances
Minimizing Node Cost : minimizes the cost of using the nodes or activating them
Minimizing Arc Cost : minimizes the cost of using the arcs or activating them
Multicast Communication : the article supports multicast requests
Hybrid : the article evolves in a hybrid context
Microservice : the article evolves in a microservice context

To begin this analysis, we can note that the number of articles dealing with orchestration issues is quite substantial even if the resolution approaches are quite similar in many of the papers. Concerning the modelling of the various problems, they are carried out through ILP/MILP which are globally easy to assimilate. Through this table, we noticed that the papers produced for the VNF-P, VNF-C and VNF-S issues take mostly into account a significant number of constraints in their work without considering the problem as whole, which would be closer to the real situations. We also note that conflicting constraints are often not taken into account. Similarly, for objective functions we notice that a significant number of works takes into account at least two objectives in their approaches. Most often the objectives are those minimizing the cost of links, as well as VNF instances. In other words, the papers try to minimize the total cost of installation of VNFs, minimize the cost of node usages, and also minimize the length of paths taken by SFC which in addition to lowering the costs of using the nodes will contribute to improve the quality of service in general.

Concerning the methods of resolution, we noticed that the vast majority follow heuristic approaches that are more realistic, as explained before. In particular, only three contributions deal with exact approaches only. For the rest, the proposals deal with heuristic coupled with exact approaches. A relevant remark is that the proposals dealing with orchestration issues in a hybrid environment are minimal, as well as for those managing microservices functions. This latter point will require a deeper investigation of the related work which will be conducted in the subsequent work of the project. Moreover, if we focus on contributions dealing with orchestration issues in hybrid environments by taking into account virtual network functions with a microservice architecture, we noticed that none of the articles meet these criteria at the same time. Besides, none of the considered papers deal at the same time with the VNF-P and VNF-C issues, which would correspond to the VNF-PR issue, in both a hybrid and a microservice context.

Considering this with respect to the MOSAICO project, we conclude that it is interesting to address the problems of orchestration by taking into account the hybrid dimension and the heterogeneous features of execution environments, as well as microservices architectures for the VNFs, and treating the placement and routing issues. The use of a heuristic approach will certainly be required, except for experiments and tests on small instances. Moreover, to be as close as possible to the reality, the number of constraints taken into account is likely to be high. Finally, regarding the security aspects of communications, which stands for one the main project's focuses, needs to be considered, especially as a conflicting constraint with those of performance.

5 Threats altering latency

Latency reduction has given raise to a large set of enhancements focusing on different components such as AQM or CCA, as presented in Section 4.1, and experimental studies whose purpose is to assess the performance of these proposals under various traffic conditions. Some anomalies have been identified such as the bandwidth starvation in case of concurrent CCA sharing the same AQM or the resilience to traffic bursts. These works led to the subsequent proposal of means to mitigate them (traffic policing or shaping, design of separated queues, TCP pacing, per flow fair queuing, etc.). However, the capability for an attacker to leverage some of these breaches to deliberately degrade the quality of other flows, or gather more resources than those expected in a fair usage of protocols, has not been studied to date in the context of low-latency solutions, and in particular by the IETF L4S. As such, in this Section we review some of the attacks that have been identified in the literature and we also highlight the weaknesses that have been already identified for the L4S solution and that would need further investigation. Given the global closed-loop system formed by a CCA and AQM in order to regulate the emission rate, we especially focus our review on congestion signaling mechanisms (e.g. acknowledgements and explicit congestion notification) that are the main entry points for an attacker located at the border of the Internet.

5.1 Scope of interest

To the best of our knowledge, latency increase has never been considered as a primary target for an attacker and is mostly a side-effect. However, it takes part of the set of features that can be considered in a RoQ (Reduction of Quality) attack [80]. It also can be seen as a side-effect of all DoS (Denial of Service) related attacks based on packet flooding since these attacks fills the queues in the upstream routers, thus leading to a latency increase for all other traffic sharing the same path.

Considering the global process of an attack, one can distinguish four general steps [23] that are: 1) information gathering, 2) assessing vulnerability, 3) launching attack, and 4) cleaning up. Each of these steps are important. However, depending on the type of the intended attack, some of them will require a closer attention as compared to others. For example, for cache poisoning or network rootkits [159], the cleaning step is way more important than it is for an ECN dissimulation [102, 97]. Besides, network attacks can take many forms [87] but we will focus on those targeting low-latency services. Distributed attacks such as distributed opt-ack [173] or distributed Low-rate DoS (LDoS) [101] are difficult to thwart since numerous compromised hosts, or sometimes partner hosts, are used to perform the attack, masking the original source of the attack.

An adapted means to implement such a phenomena consists of generating misbehaving flows, which include unresponsive flows and malformed flows from non-compliant entities. A misbehavior can be accidental, for example, due to a homemade implementation of a TCP stack, or intentional. A legitimate node or end-system can also be manipulated and thus propagate a misbehaving flow. Another formulation in relation with manipulation attacks is protocol "gullibility" [178], that is to say, "*the ability of some of the participants to subvert the protocol without the knowledge of the others*". As pointed out in [97], a manipulation attack has three main characteristics:

1. Attackers induce the legitimate participants to change behavior to their own advantage or to degrade network quality of legitimate participants.
2. The induced behavior is actually protocol-compliant under some circumstances and network conditions.
3. A single manipulative act alone is usually not enough to perform an attack. The manipulation must be repeated several times in order to obtain the desired impact.

Issues addressed		Constraints						Solving approach		Objectives			Other features		
	Articles	Precedence	Function Capacity	Node Capacity	Arc Capacity	Conflict	Latency	Exact Method	Heuristic Method	Minimizing Function Cost	Minimizing Node Cost	Minimizing Arc Cost	Multicast Communication	Hybrid	Microservice
VNF-P	[131]	X	X	X	X		X	X			X			X	
	[184]	X	X	X	X		X	X	X	X	X	X			
	[58]		X	X					X	X	X				
	[157]	X		X	X		X		X		X				
	[152]		X	X	X		X	X	X		X				
	[11]		X	X	X		X		X		X			X	
	[103]				X		X			X		X		X	
	[88]	X		X	X				X	X	X			X	
	[112]	X	X	X	X		X		X	X		X		X	
VNF-C	[161]	X	X	X	X		X	X	X	X	X	X			
	[113]	X		X	X		X		X						
	[110]	X		X			X		X		X				
VNF-S	[156]	X	X	X			X	X							
	[108]	X	X	X	X		X		X	X	X	X			
	[109]	X	X	X			X		X						
	[149]	X	X	X					X						
VNF-PS	[130]	X	X	X			X	X	X	X	X	X			
VNF-PR	[18]	X	X	X	X		X	X	X		X	X			
	[79]	X		X	X		X		X			X			
	[117]		X	X		X	X		X	X					
	[7]		X	X	X			X			X				
	[81]	X		X	X				X			X			
	[118]	X		X	X		X		X		X	X			
	[6]		X	X			X		X		X	X			
	[14]	X	X	X		X	X		X	X		X			
	[194]	X	X	X	X		X		X	X		X	X		
	[95]	X	X	X	X		X		X	X		X	X		
	[13]	X	X	X			X		X	X		X	X		
	[155]			X	X		X		X	X	X	X	X		X

Table 3: Summary table of orchestration issues

These anomalies and attacks can significantly cause service degradation by lowering the performance, bandwidth and security, or by increasing end-to-end latency. In this Section, we will concentrate on this latter case and especially on attacks that are likely to increase the Bandwidth-Delay Product (BDP [183]).

5.2 Classification of known attacks threatening latency

One can identify three main classes of attacks that may be a threat to latency that are described in the following sub-sections.

5.2.1 Hacked ACK

This attack can be used to manipulate an endpoint in a TCP communication (usually the sender) or to create congestion in intermediate nodes (usually the edge router or an access router shared by targeted victims). An optimistic acknowledgment (*opt-ack*) attack [173, 97], for instance, consists of sending an ACK before actually receiving the data. To be more subtle, an attacker can send opt-ack for any segment that it actually receives, concealing any packet loss. This is called the *lazy opt-ack* [173] [4]. The distributed variant also exists and is called *distributed opt-ack* attack, and requires multiple attackers to be coordinated to target different set of victims. The intended effect is to saturate the paths between many targets and the misbehaving receiver(s), eventually provoking collateral damages on legitimate users not initially targeted by the attacker. A misbehaving receiver can manipulate acknowledgments in another way: by artificially subdividing the acknowledgment of a group of segments into several acknowledgments of several smaller groups for the same data. This is called the *ack-division attack* [163]. However, this has normally been fixed in RFC5681[26] that recommends counting acknowledged bytes instead of counting acknowledged packets. Finally, another network attack which uses acknowledgment is the *ack-storm DoS* attack [2]. It is not really effective in increasing the latency as it provokes an infinite loop between a server and a legitimate client. Besides the attacks implemented over standard TCP implementations, other transport protocols can also be leveraged such as those implemented in a peer-to-peer P2P file sharing context. Especially, micro Transport Protocol (μTP) has been designed to make BitTorrent more Internet Service Provider (ISP) friendly. It is based on UDP with a congestion control termed Low Extra Delay Background Transport (LEDBAT) which assumes cooperation of the receivers. Authors of [4] have identified two attack scenarios: congestion collapse and bandwidth stealing. The former fits in the scope of threats to low latency, and effects can be packet loss, increase of queuing delay and blocking of new connections. As pointed in the paper, the peer-to-peer context requires trust of all participants, not only endpoints (if we assume that intermediate routers and ISP middleboxes are considered trusted). This means that an attacker could perform MitM attacks more easily and introduce additional delay in the targeted path by, for instance, lying about the delay measurements. According to RFC 6817[170]: *LEDBAT requires that each data segment carries a "timestamp" from the sender, based on which the receiver computes the one-way delay from the sender and sends this computed value back to the sender*) or performing a opt-ack attack or a lazy opt-ack attack.

5.2.2 Hacked ECN

This attack can be used to manipulate an endpoint of a TCP communication (usually the sender) or to create congestion in intermediate nodes using several techniques: an attacker may lie saying that its emission rate is reduced while actually keeping the same rate, an attacker may conceal a congestion notification by erasing the flag, or an attacker may lie by sending a false congestion notification in order to steal more bandwidth. The concealing attack is demonstrated in [64] and replayed in [97] where a method, based on a combination of static analysis with symbolic execution and dynamic analysis with concrete execution, is developed to help discover manipulation attacks in TCP, SCTP, 802.11 MAC and ECN. In [102], these three categories are detailed and an approach transforming protocol specification into using an Extended Finite State Machine and P4 [31] network programming language is presented. Finally, [178] introduces the notion of protocol gullibility and applies it to ECN in a game theory context with honest players and manipulative players.

5.2.3 Low-rate DoS

The general model for LDoS attacks is described in [197] and [124], and is represented in Figure 23. To be effective, burst of packets should be sent in a periodic way to overflow the router's queue and cause packet loss of legitimate users. An attacker can target the victim's Retransmission Timeout (RTO) by synchronizing the bursts with the victim's RTO duration. In this way, he/she could sustain the DoS while causing congestion in intermediate routers and, thus, provoke collateral damages by unintentionally performing a DoS attack on other legitimate users. Bursts last L amount of time for a magnitude of R, and each burst should follow a period of T, where T is adjusted with a distant RTO.

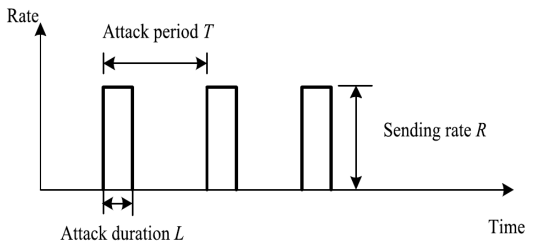


Figure 23: The general model for LDoS attacks [197]

5.3 L4S known vulnerabilities

L4S has been developed by the Reducing Internet Transport Latency (RITE) project[4] (ICT-317700) which is part-funded by the European Community under its Seventh Framework Program. It was evaluated in several networking conditions, specifically under overloading conditions with various traffic classes [179]. Its general architecture is under standardization by the IETF [38]. Its scalable congestion control, Prague[166], and its dual queue coupled active queue management algorithms, [165] (DualQ coupled AQM) can be implemented by the DualPI^2 [12] algorithm or by the Curvy RED algorithm.

5.3.1 Abnormal traffic support

Even with a well-designed traffic filtering, traffic policing, or per-flow traffic scheduling, the core AQM of L4S, named DualQ Coupled AQM, needs to anticipate misbehaving flows or even legitimate congestion due to overloading, as addressed in the associated draft [165] and also in the IETF AQM recommandations [16] which states in its security dedicated section that *many deployed network devices use queueing methods that allow unresponsive traffic to capture network capacity, denying access toother traffic flows. This could potentially be used as a denial-of-service attack. This threat could be reduced in network devices that deploy AQM or some form of scheduling. We note, however, that a denial-of-service attack that results in unresponsive traffic flows may be indistinguishable from other traffic flows (e.g., tunnels carrying aggregates of short flows, high-rate isochronous applications). New methods therefore may remain vulnerable, and this document recommends that ongoing research consider ways to mitigate such attacks.*

Under overload, a preoccupation of L4S is to sacrifice some performance aspect in order to preserve low-latency services. Potential solutions are provided in [165], each of them reducing different aspects (throughput, delay, drop). As pointed out, *these choices need to be made either by the developer or by operator policy, rather than by the IETF.* Especially, to avoid starvation in the classic queue in overloaded conditions, L4S queue needs to sacrifice throughput or delay. The former solution can be performed with a well-adjusted weighted round robin. The sacrifice of L4S delay can be performed by ”leaking” some delay from classic queue to L4S queue with the help of a time-shifted FIFO scheduler (which is the solution used in the current L4S implementation). The time shift can be adjusted when the router is experiencing overload.

Another vulnerability comes from the congestion signal saturation that can occur when unresponsive flows are too demanding. In this context, saturation could be alleviated by performing some drops in the L4S queue. This requires introducing some thresholds for ECN marking, or allowing delay in both L4S and non-L4S queues, which could lead to simply tail drops since the current L4S implementation adopts the ”drop on saturation” policy.

To start quantifying and understanding the impact on latency and throughput during overload conditions and congestion signal saturation, some early tests have been realized that are presented in [179]. Also, in [30], researchers focused on the reproducibility of the results from the original paper, to evaluate the co-existence in DualPI^2 of scalable CCA implementations, namely DCTCP and TCP Prague for low-latency traffic, and classic CCA implementations (here, TCP Cubic). They showed that window

[4] https://riteproject.eu/

fairness was not always respected with DCTCP. However, unlike TCP Prague, DCTCP is not intended to be deployed over the Internet, thus making this insight not actually an issue. Yet, sensitivity to packet bursts has been identified in L4S and needs to be further tested and measured. Another point of interest indicated in this paper is the necessity of reproducibility. For instance, the Linux kernel has evolved, modules and network stack implementations have been updated since the original seployment, making it difficult to compare and properly test latency and L4S, and requiring further research to improve this situation.

In [134], the authors show that L4S schedulers are not always able to handle RTTs heterogeneity between classic traffic and L4S traffic. They propose a Virtual-Dual Queue Core-Stateless AQM (VDQ-CSAQM) that fits with L4S requirements. This novel AQM is evaluated with a DPDK-based testbed.

All these studies demonstrate that if L4S provides a relevant solution to ensure the coexistence of both classic and low-latency traffic over the Internet, several situations, such as event induced by legitimate behaviors of endpoints, require a careful study and, particularly, those related to overloading, traffic bursts, and fairness related to different congestion control algorithms.

5.3.2 Potential protection mechanisms

In order to deal with the aforementioned issues identified and anticipated by both IETF and the RITE project, the L4S architecture draft [38] specifically addresses situations related to overloading, as well as traffic rate policing and burst policing[5].

The authors of the draft expose various arrangements to address these questions, even if none of the queue protection capabilities are considered as necessary for the L4S architecture:

- Local bottleneck queue protection: prevents low-latency queue to be filled with queue-building flows that may accidentally or maliciously classify themselves as low-latency flows. This protection is based on scoring flows in relation with their contribution to queuing delay [37].
- Distributed traffic scrubbing: allows re-routing of malicious flows in dedicated scrubbing facilities.
- Local bottleneck per-flow scheduling: isolates non-bursty flows from bursty. Per-flow scheduling shows more potential that per-flow policing (see discussion in Section 5.2 in [38]).
- Distributed access subnet queue protection: a per-flow queue protection that consists in a queue structure distributed across a subnet inter-communicating using lower layer control messages.
- Distributed Congestion Exposure to Ingress Policers: The Congestion Exposure (ConEx) architecture [125] uses egress audit to motivate senders to truthfully signal path congestion in-band where it can be used by ingress policers. An edge-to-edge variant of this architecture is also possible.
- Distributed Domain-edge traffic conditioning: An architecture similar to DiffServ[25] may be preferred, when traffic is proactively conditioned when entering into a domain, rather than reactively policed only when it leads to queuing.
- Distributed core network queue protection: The policing function could be divided between per-flow mechanisms at the network ingress that characterize the burstiness of each flow into a signal carried with the traffic, and per-class mechanisms at bottlenecks that act on these signals if queuing actually occurs once the traffic converges. This is an idea similar to core stateless fair queuing [134].

Misbehaving flows that exploit hacked ECN attacks are also partially anticipated. Various RFC have been published in order to design a more robust ECN with an addition of security options [73, 189, 24], even regarding a combination with Active Queue Management algorithms [16]. But there is no concrete merging of these ideas and some solutions tend to make others unnecessary or are contradictory. Yet, L4S-specific solutions are still actively discussed in the draft [164].

All these studies anticipate the need for L4S to better handle situations in which the traffic is not compliant with standard expectations. Especially, the security has not been considered as a strong requirement for the L4S architecture. Early measurement campaigns first highlight some limits in preventing either flow starvation or traffic bursts. If all these studies have been achieved with the idea of verifying the robustness of the L4S architecture, they have restricted their scope to legitimate behavior of endpoints. However, the implementation of alternate behaviors in which an attacker aims at deliberately exploiting these flaws has not been explored to date.

[5]The L4S service relies on self-constraint - limiting rate in response to congestion as well as self constraint in terms of limiting latency (burstiness). Whether burst policing is necessary is still in discussion.

6 Candidate technologies solutions for hosting network modules

This section describes the technologies we envision to use for offering a secure and efficient delivery of LL applications within the MOSAICO project. First, we present our choice for the programmable control plane, then the one for the programmable data plane.

6.1 Background on network function virtualisation

The network function virtualization is a set of technologies taking its origin in cloud computing and whose purpose is to leverage the power of virtualization to networking. In order to clearly understand the area encompassed by this paradigm, we provide in the following some definitions that are relevant in the context of the MOSAICO project. More specifically, the concept of network service orchestration, is defined as a set of coordinated processes, allowing the automation of the management and control of the information system in order to achieve a common goal [188].

There are different and complementary definitions in the literature. For example The National Institute of Standards and Technology (NIST) [29] was among the first to define the orchestration of network services. According to NIST, *orchestration is the process of managing, coordinating virtualized infrastructure to deliver network cloud services to customers*. Also, The Open Networking Foundation (ONF) [63] also formulated the definition of orchestration as *the use and selection of resources by the Orchestrator in order to meet client demands*. The ONF also defines network service orchestration as *a feature of the SDN controller and states that the main features of orchestration are: the division of large service requests into service components and then the sharing of these components between supported platforms*.

In addition to these definitions proposed by organizations, we find certain definitions in research papers. For example, for [177] the network service orchestration allows *programmability for end-to-end service creation and deployment and dynamic control of the network through a single interface*. Thyagaturu et al. in [129] define orchestration as the *coordination of service functions and operations in an upper layer creating an abstraction of the physical lower layer. In [150] orchestration was defined as the automatic management of complex and service systems*.

The main essential concepts related to NFV are as follows [51, 138]:

- Network Function (NF): represents a functional block within a network infrastructure that has well-defined interfaces and behaviors. We can take a firewall, load-balancer, Deep Packet Inspection (DPI) as illustrative examples.
- Virtualized Network Function (VNF): represents a software NF that can be deployed on a virtualization infrastructure.
- Network Service (NS): represents a chain of NF/VNFs in a well-defined order provided in a service specification and that can be implemented in the NFVI.
- VNF Forwarding Graph (VNF-FG): represents a graph standing for the logical links between the different NF nodes. This allows to describe the traffic flows between these different nodes.

Additionally, since VNFs can be dynamically deployed and re-deployed on different physical and virtual resources of the NFV infrastructure, the NFV architecture must implement three main components that are :

- NFVI: NFV Infrastructure consists of nodes that support physical and virtual NFs (NFVI-PoP) and allow their execution. The NFV Infrastructure interconnects NFVI-PoP as well as IT and storage resources.
- NFV-MANO: The NFV Management and Orchestration covers the orchestration and lifecycle management of physical and/or software resources that enable infrastructure virtualization. It also allows the life cycle management of VNF. NFV-MANO can be divided into three entities which are:

  VIM: Virtualized Infrastructure Manager to control and manage NFVI computing, storage and networking resources.

  VNFM: VNF Manager is responsible for managing VNF

  NFVO: Responsible for the integration of new Network Services (NS) and Virtual Network Functions (VNF) packages; NS lifecycle management and overall resource management;

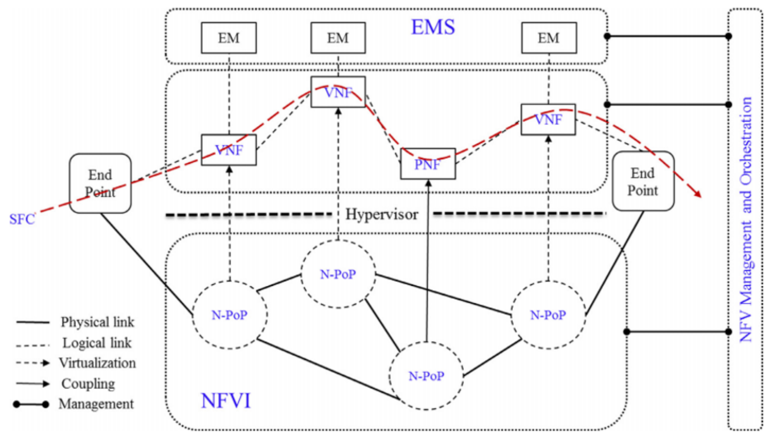


Figure 24: NFV architecture [65]

- Element Management System (EMS): an EMS is a set of elements (EMs) responsible for managing the instantiation of VNFs, their executions and their deployments.
- Network Point of Presence (N-PoP): represents the locations in the network or can be instantiated virtual or physical network functions.

Since its initial proposal in 2013, NFV is still in evolution and there are some areas which concentrate some research efforts. These are:

Resource Management: Resource management is one of the different strategic challenges in orchestrating VNF. The servers that allow their execution have limited storage and calculation capacities. Therefore, in order to optimize the consumption of resources without compromising the quality of service, it is essential to properly manage these resources. In this context, three main challenges are defined by [192] that are: The PoP NFV localization, function placement and dynamic resource management.

Distributed Management: Current MANO approaches focus primarily on centralized solutions, which implies limitations in scalability. The reason is the overhead costs of communication and the delay in collecting and analyzing data from a large number of heterogeneous sources. Therefore, in order to better manage the dynamism of requests, it is necessary to put in place monitoring mechanisms to feed information to the management entities in order to make the necessary configuration changes.

Security: SDN and NFV focus on remote programmability of network resources and functions. This opens the door to new attacks and threats that could have a devastating impact as compared to a conventional non-NFV environments. Among the different solutions studied only CloudBand includes a security solution allowing the prediction of anomalies. Therefore, new safety concerns in the multi-vendor environment arise and need to be addressed.

6.2 Microservices technologies

In [62], microservices are defined as follows: *A microservice is a cohesive, independent process interacting via messages.* This notion is an architectural style that come from Service-Oriented Computing and tends to be transposed to the networking area. It can be seen as the evolution of Oriented-Object Programming and design patterns in a context of decentralization.

Decomposing a monolithic architecture into a microservice-based one can solve several issues and bring dynamism, modularity and code reuse in distributed environments. A microservice architecture shows benefits for continuous integration via gradual transitions, without having to reboot the whole system when adding new features. Each microservice is supposed to be built with small amount of code, thus, the scope of a bug is limited, the isolation is improved by design, and upgrades can be deployed independently minimizing downtimes. Scaling is also improved since bottlenecks or resource intensive components of an application can be more easily identified and specifically scaled up without changing other components or duplicating the whole application.

However, as pointed out in [171], an important constraint should be considered when designing architectures with a microservice-based approaches. In order to guarantee inter-operability in a context of huge technical heterogeneity, two key points are:

- the design of inter-microservice communication
- microservice management

The first key point can bring too much communication overheads that might lead to loose the performance benefits as compared to a monolithic approach. The second one is more significant for microservices and involve two main paradigms: choregraphy and orchestration.

Choregraphy is a decentralized approach for controlling and monitoring and often using event-based and publish/subscribe mechanisms to make microservices cooperate together while keeping them independent. Behavioural specifications and behavioural programming of microservices constitute a way to design a choregraphy but these techniques still are at an early stage of maturity. Another way is to make microservices learn independently in order to make them rapidly adapt to the real situations in a non-deterministic manner. This can be done by leveraging machine learning. This architectural style for management brings flexibility and lets information propagate gradually among microservices but is more complex to develop.

Orchestration, on the other hand, uses a centralized approach and is widely adopted because it brings more control and makes monitoring easier for obtaining a quick overview of the whole network activity. This approach require one or several central units that oversee the whole infrastructure to coordinate the microservices. Control communications often rely on REST[70] or other client/server structures, scheduling and queuing mechanisms are usually conjointly used. This architectural style for management is easier to understand and develop, that contributes to its popularity and its wider adoption compared to choregraphy. But if this organization is not properly designed, it can lead to unequally distributed responsibilities among services and thus give some more centralized control than others which can be a threat for privacy and resiliency, and is not compatible with the microservice approach.

6.2.1 Microservices architectures

We identify that microservices concept can have different meanings and be applied at different levels. Yet, microservices tend to be transposed to the NFV ecosystem. We summarize in this section the different options for introducing microservices in this context.

As surveyed in [53], microservice-based architectures are increasingly designed. We can classify microservice architectures into two categories: OpenNetVM-based and Click-based but several projects at various states of maturity and openness are based on OpenNetVM. Yet, there are some attempts to explore other paths such as in NetBricks [145] where the authors avoid virtualization in order to improve the performance and consider another way to ensure packet isolation with a memory-safe language (Rust). We can notice that most of these NFV implementations integrating microservices are using DPDK [74], a software solution that reduces data copying by providing *"network interface controller polling-mode drivers for offloading TCP packet processing from the operating system kernel to processes running in user space"*.

Among them, one of the solutions that is most interesting for MOSAICO, is OpenNetVM [196], a high-performance NFV platform based on DPDK [74] where network functions can be isolated using Docker containers. It is often cited or even used as a starting point before adding features. Another technology, Click [96], is often cited or used in papers, such as in DockNet[119], OpenBox [33], CoMb [169] or more recently µNF [54].

Containerization and virtualization present benefits for flexibility and management, but may incur severe performance penalties due to frequent context switching, especially for simple network functions. For complex ones, the processing time of the NF usually dominates all other factors.

NetBricks[145] proposes a programming model and a safe runtime system relying on Zero-Copy Software Isolation using a memory-safe language (Rust) that ensures packet isolation within a shared memory. NetBricks runs as a single process which may be assigned to one or more cores for processing, and one or more NICs for packet I/O. NFs are built using NetBricks' abstraction and are composed of user-supplied functions. The authors have developed 5 programming abstractions: packet processing, bytestream processing, control flow, state management and event scheduling. NetBricks' runtime environment is responsible for providing NF placement, scheduling and inter-NF isolation. Packet forwarding between NFs is made using function calls enabling low-latency processing.

Few functional prototypes are completely developed, and DPDK alone lacks a higher level of abstraction.

OpenNetVM is a platform for high performance network service chains. It can run several NFs in Docker containers or as processes and uses DPDK for high performance I/O. OpenNetVM is composed of NFs and a Manager that uses 3 cores (one for NIC's RX, one for NIC's TX and one for statistics and flow management). Incoming packets are stored in huge pages by DPDK, allowing the NFs to access them. The NF manager maintains a flow table thanks to its flow director. The flow director is responsible for mapping packet flows to the appropriate service chain. Each NF is identified by its ServiceID and its InstanceID, several NFs can share the same ServiceID but the InstanceIDs are unique. For a given ServiceID, the manager will decide which instantiated NF will process the incoming packet, improving flexibility and fault tolerance. In case of a sudden freeze of an NF, the manager can simply route the packets to another one using the same ServiceID. Figure 25 illustrates the architecture.

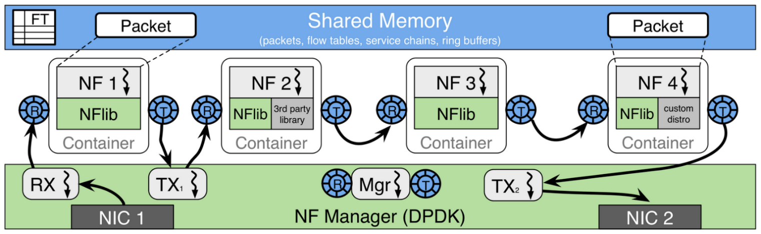


Figure 25: OpenNetVM architecture

OpenNetVM is built on top of DPDK, thus it benefits fully advantages from DPDK. The *zero-copy* allows to avoid overhead of OS kernel packet processing, such as, copying packet, OS interrupt. Using hugepage for memory management avoids overhead paging and sharing among NFs allowing them to access directly without address translation. DPDK increases performance by processing packets by bulk. It provides also a high performance Cuckoo hashtable to identify quickly packet flows and NFs for easily distributing packets among the NFs. OpenNetVM brings also its own advantages by dynamically managing and reconfiguring the NFs and their chains at runtime. It supports the uses containers to run NFs, thus providing benefits, such as: the use of NFs from different vendors that can easily coexist while retaining isolation; each container can include its own libraries and dependencies; and resources can be precisely allocated. Finally it is opensource under BSD license.

Beside that OpenNetVM has also its limitations. The number of NFs is limited by the number of CPU cores running DPDK since each NF is attribute a single core. Furthermore, because it heavily depends on DPDK, all NFs should be on the same CPU, thus same host. It loses its advantages when running NFs on different CPUs. Although OpenNetVM supports a NF that is deployed inside a containers, the NF does not total independent. It must use OpenNetVM's NFlib API for registering when starting, registering a callback to get packets, and deregistering before ending.

NF scaling is usually made by running an additional thread but it can be made by dedicating an additional core. From the Git repository [167], we can see that an experimental integration of a semaphore-based communication system, that allows multiple NFs to run on a shared core through a interrupt-based technique driven by the NF Manager, is already available. This is based on the hybrid-polling model outlined in Flurries[195] and further enhanced in NFVnice [98] and REINFORCE[99].

NFV deployment is not often customizable, granularity of network functions is usually coarse-grained and few works address a flow-centric approach. Flurries [195] proposes a fine-grained NFs for flexible per-flow customization leveraging a pool of Docker containers embedding pre-configured NFs which are orchestrated with a OpenNetVM-based platform and, thus, rely on DPDK. Network functions have a shared memory for improving packet processing efficiency. A hybrid architecture that uses polling to rapidly handle packets from the NIC and interrupt-based NF processing to allow a large number of functions to share CPU core has been designed. Flurries also provides an adaptive scheduler that dynamically adjusts packet batching to improve latency. Each NF is isolated within a Docker container and only packets are shared in memory, allowing zero-copy I/O. The Flurries architecture is represented in Figure 26.

When a new packet arrives, the flow lookup table either redirects it to an existing flow or sends it to the flow director which can be interconnected with a SDN controller for SFC placement and is in charge of assigning an idle NF to new flows. Each NF is aware of its flow duration and is able to alert the CPUs

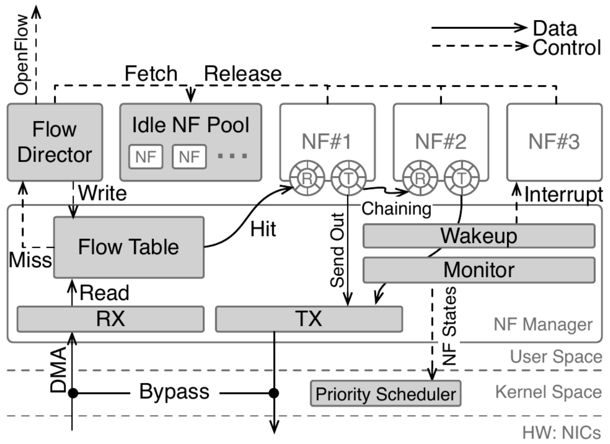
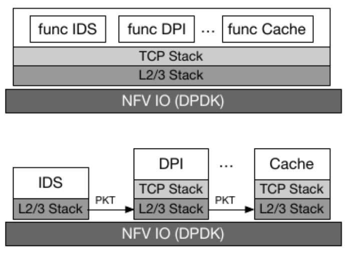


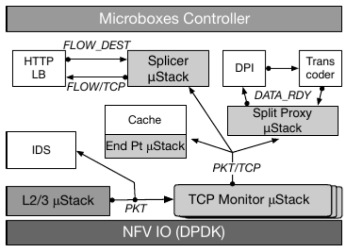
Figure 26: Flurries architecture

when they go idle. The wake-up system alerts the NFs when a new packet is ready for processing. The wake-up system consists in a state machine composed of a CPU Scheduler block list (representing idle or asleep containers) and a CPU scheduler run list (representing active or running containers). The scheduler prioritizes flows depending on different QoS requirements, ensuring flow-isolation. This priority is fixed when the flow is instantiated. A dynamic threshold triggers state transitions depending on the number of packets available for batching. This threshold is adjusted via an Additive Increase Multiplicative Decrease (AIMD) mechanism and a timer.

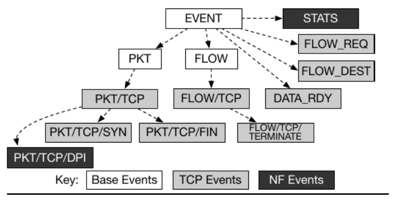
Most of service chaining policies are packet-centric, which is not efficient enough for complex NFs. Microboxes, an OpenNetVM-based platform, suggests an abstraction for service chaining based on DPDK, thus, enabling zero-copy, implementing a TCP Stack (μStacks) and a publish-subscribe-based API for event management (μEvents). μEvents provide a high-level interface based on event flows rather than packet flows. A controller is in charge of event subscriber management. For each event, a single data structure is allocated. This data structure contains metadata, a snapshot of the stack state at the moment of allocation, and a frontier pointer that guarantees stack consistency among parallel treatments. μStacks provides a way to eliminate redundant calculations about stacks and protocols in SFCs. Customizations can be made in order to have an asynchronous parallel NF processing for a better scalability. The main particularity of Microboxes is the decoupling of stack processing from NF processing. After the treatment of the service chain, TCP reconstruction is operated and the packet is sent back to the NIC.



(a) Monolithic (top) and pipelined architecture (down)



(b) Microboxes architecture



(c) Microboxes event types

Figure 27: (a) Monolithic architectures combine NFs as function calls in a single process, but this hurts deployment. Pipelined NFs deployed as containers are easier to manage, but prone to redundant computation. (b-c) Microboxes customizes protocol processing in μStack modules and facilitates communication with a hierarchy of μEvent types.

NFVnice [98] introduces a rate proportional scheduling for NFV service chains in OpenNetVM. It leverages `cgroups`, a user space scheduling abstraction offered by the operating system that let NFVnice automatically weights process priorities to control when each network function is scheduled, without changing anything at the operating system level, and regardless of the kind of OS's scheduler. It weights NFs based on their arrival rate and the required computation cost.

When service chains are long and busy, bottlenecks can occur and some NFs may drop packets already processed by upstream NFs. To fix this, NFVnice adopts a dynamic backpressure mecanism as a service chain level congestion control that prevents this kind of wasted work. The NF manager is able to detect overloaded NFs and applies back pressure upstream at the flow-level, which minimizes impacts on the overall performances as compared to whether it was applied at the NF level.

High availability for NFV is a feature that can imply severe performance degradation when requirements for QoS are high. REINFORCE[99] is a framework based on OpenNetVM that guarantees efficient resiliency for NFs and service chains by replicating some functions in standby NFs (remotely or locally) and capturing packets in circular buffers. In this way it can rapidly recover from failures.

ClickOS is a high-performance virtualized software middlebox platform enabling NFV. It is based on Click [96], a software architecture for building configurable routers with independent packet processing modules called elements. ClickOS extends this concept further for other types of middleboxes. It uses virtual machines (VMs) which are heavier than containers due to extra levels of abstraction and implementation. However, to minimize performance overheads on throughput and latency in the heavy VM-based virtualization, ClickOS uses para-virtualization, meaning that some changes have to be made on the guest OS. ClickOS improves efficiency of packet processing, but lacks network-wide control.

OpenBox[33] is a software-defined framework for developing, deploying and managing network functions. It is based on the Click modular router. It mainly provides the possibility of function redundancy by factorizing common subfunctions in a service chain thanks to a graph merging algorithm. OpenBox consists of three types of abstractions: (1) user-defined OpenBox Applications (OBA) that provide NF specifications; (2) OpenBox controller (OBC) that is able to merge processing graphs from multiple NFs and can be connected with a SDN controller; and, (3) OpenBox instances (OBI) that directly compose the data plane and receive a processing graph from the controller. OBIs are implemented with the Click modular router. OBC and OBA communicate through a REST API, and OBC and OBI communicate through a REST channel over HTTPS with JSON-encoded messages. With this organization, the total decoupling between the data and control planes is assured.

Middleboxes are specialized network appliances that modify, inspect, filter and manipulate traffic for purposes other than packet forwarding. They usually are closed systems that are independently deployed, blind to other middleboxes in their neighbourhood.

Consolidating Microboxes sets out a goam that gives it its name, CoMb[169], an architecture for middlebox deployments that focuses on opportunities for consolidation. "De-specialization" of middlebox infrastructures brings greater flexibility, but can lead to lower resource utilization. However, the modularity it offers can be opportunistically used to consolidate the network and thus improving hardware resource efficiency in the end. Consolidation can also exploit spatial distribution of middlebox to balance the workload among different locations. In order to do this, a controller for resource management is required. CoMb plays this role, using the Click modular router running on general purpose hardware. Resource consolidation is made using application multiplexing, software element reuse and spatial distribution. It provides new ways for resource savings.

Some works related to NFV are replacing monolithic hardware with monolithic VNFs. It is a good point for decoupling hardware from network functionality, but when chaining functions together it can result in subfunction redundancies, as highlighted by OpenBox. µNF[54] proposes re-architecturing the NFV ecosystem with a greater VNF modularity and service composition flexibility and achieves the same throughput as monolithic VNF based SFCs by using an average of less CPU cycles per packet. µNF is a disaggregated packet processing architecture inspired by CoMb that also uses the Click modular router. It leverages software reusability with an architecture designed for composing lightweight VNFs and SFCs via loosely coupled components called µNFs that can communicate with each other within a shared memory pool. This is done using DPDK with each µNF being a DPDK process written in C++ orchestrated by a controller written in Python.

In [90], the authors present an architecture for CDN (**C**ontent **D**elivery **N**etwork). The CDN components are divided into microservices, distinguishing the '*Data plane*' and the '*Control plane*'. Both are composed of several domains.

- The Data plane represents the resources needed. It is composed of two elements: the Provider Domain and the Content Provider Domain. The Provider Domain is the PoD (**P**oint **o**f **D**eployment). It is composed of a deployment agent.
  The Content Provider Domain is a media server. It contains all the resources that need to be replicated in the newly-deployed surrogate server.

- The Control Plane must manage the components and handle the signaling. It is also composed of

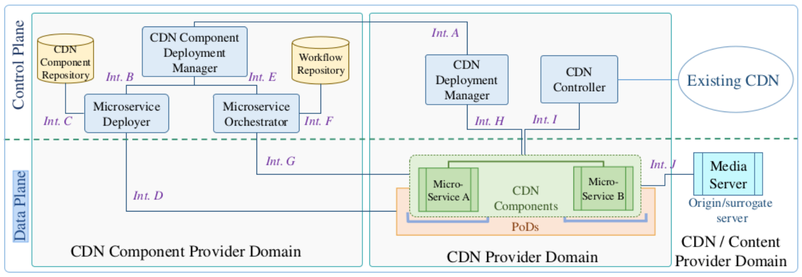


Figure 28: High-level architecture [90]

a Provider Domain and a Content Provider Domain.
The Provider Domain consists of two components. The first one, the CDN Controler, must redirect users to the appropriate server. The second, the CDN Deployment Manager, is in charge of choosing the PoD closest to the users.
The Content Provider Domain consists of three components. There is: the CDN Component Deployment Manager that must decompose a request into a set of microservices; the Microservice Deployer whose role is to deploy the microservices on the PoD; and the Microservice Orchestrator that orchestrates the microservices and makes them communicate with each other.

The authors measure the time taken in each step of the process. To begin with, there is the '*Deployment Delay*'. This step lasts from the deployment request ('*CDN Component Deployment Manager*') to its acknowledgement from the '*Microservice Deployer*'.
Then, there is the '*Orchestration Delay*'. This stage begin at the time of the orchestration request ('*CDN Component Deployment Manager*'). It ends when the '*Microservice Orchestrator*' makes its acknowledgement to the '*CDN Component Deployment Manager*'.
Finally, the '*Provisionning Delay*' takes place between the provisioning request ('*CDN Deployment Manager*') to the surrogate server's registration with the '*CDN Controler*'.
The authors repeated the measurements 10 times and the results are depicted in the Table 4.

	Average	Standard Dev
Deployment Delay	6.03	0.22
Orchestration Delay	7.97	1.08
Provisionning Delay	19.85	1.18

Table 4: Results [90]

As future work, the authors plan to use '*choregraphy*' instead of orchestration.
In [55], the authors propose an architecture that decomposes network functions (NFs) into µNFs. A µNF performs a specific task on a packet. A SFC (**S**ervice **F**unction **C**hain) is a chain of NFs. In a typical architecture, some operations are repeated along a SFC. The proposed architecture allow to factorize these redundant operations. Figure 29 shows the overall architecture. It exhibits two components:

- *µNF Orchestrator*': Is responsible for the overall decisions. It generates a µNF graph and distributes it to several devices.
- '*Orchestration Agent*': Given a µNF sub-graph, it is in charge of its local deployment.

Some optimisations are performed, such as parallel execution of µNFs. In fact, consecutive µNFs that don't modify the packet or that don't write in the same region can be parallelized. The authors showed that the more a µNF is complex, the more interesting it is to parallelize it. They also showed that the latency grows linearly with the processing graph's length and with the µNF's complexity.

6.3 Programmable network equipment

Recently, the idea of being able to program the data plane (how packets are processed in the network) has emerged. The P4 (Programming Protocol-Independent Packet Processor) language[42, 142] has been designed to allow to have fully configurable network equipment. This section briefly introduces the P4 concept, before presenting some related work dealing with P4 and QoS and security, the two key topics of MOSAICO.

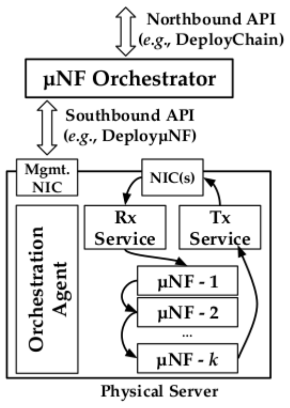


Figure 29: Architecture [55]

6.3.1 Programming Protocol-Independent Packet Processor

Most network devices, such as, routers, switches, etc. were not, at least initially, configurable. These devices were dedicated to network functions and, once deployed, only realized these. It was not possible to differently configure them or to change their behavior. For several years now, SDN (Software-Defined Networking), using the Open Flow protocol, allows programming the control plan of the network (using configuration sent from a controller). This gives an abstraction and a more global view of the network. SDN and NFV are the key enabling technologies for 5G networks [141]. These technologies along with Software-Defined Security (SDS) are essentially relying on software-based approaches and allow to significantly reduce the need to have dedicated hardware devices by introducing virtualised devices and APIs to process and control network traffic.

For few years, P4 (Programming Protocol-Independent Packet Processor) language[42, 142] has been designed to allow administrator and developers to dynamically and fully configure network programmable equipment. It introduces the ability to program the devices addressing the controller's changing requirements, rather than being constrained by a prefixed traditional switch design. With P4, it becomes easy to implement new features and to quickly deploy them at large scale. P4 is potentially becoming the disruptive technology [146] enabling the programming and customizing of the data plane of next-generation SDN/NFV-based mobile networks. It opens the door to a finer management of network packets, of queues in the network devices and allows better communication between the devices and their controller by introducing an interface between them. P4 is 'protocol independent' because it allows to quickly adapt and deploy new protocols and their management within the same program.

The P4 language is in its second version[42]. It aims at being deployed on P4 compatible network devices which can interpret a compiled P4 file, in JSON format or other. A basic Linux implementation is provided by the P4 consortium (BMv2), but some hardware manufacturers starts to provide P4 chipset, for example Tofino.

There are 4 main modules in a P4 program: Parser, Ingress, Egress and Deparser, as being depicted in Figure 30.

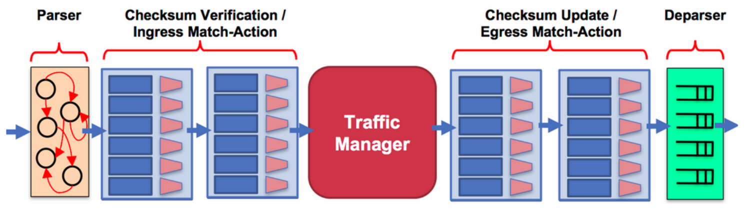
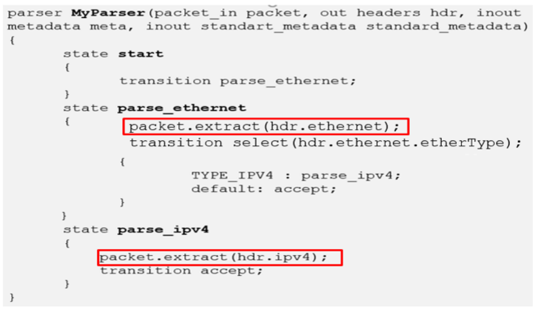

Parser
Checksum Verification / Ingress Match-Action
Traffic Manager
Checksum Update / Egress Match-Action
Deparser

Figure 30: The 4 main components of a P4 program [142]

- The *Parser* allows to select the different headers of the packets on which the program will be able to make actions (Ethernet, IPV4, UDP, ...). Figure 31 is an example on how to parse Ethernet and IPv4 headers. The fields of these headers should be declared in the P4 program or in an external file, referenced by the P4 program.

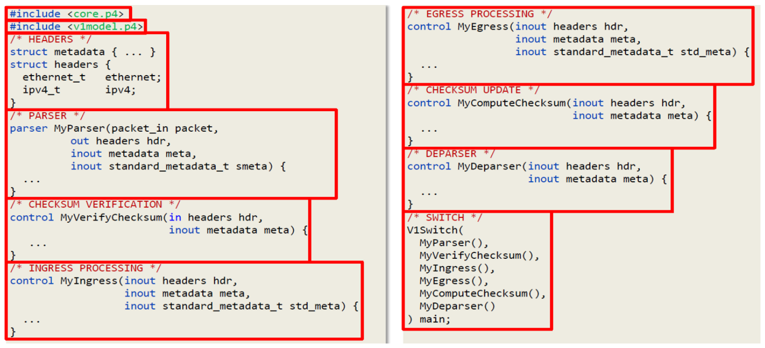


```
parser MyParser(packet_in packet, out headers hdr, inout
metadata meta, inout standart_metadata standard_metadata)
{
        state start
        {
                transition parse_ethernet;
        }
        state parse_ethernet
        {
                packet.extract(hdr.ethernet);
                transition select(hdr.ethernet.etherType);
                {
                        TYPE_IPV4 : parse_ipv4;
                        default: accept;
                }
        }
        state parse_ipv4
        {
            packet.extract(hdr.ipv4);
            transition accept;
        }
}
```

Figure 31: The parser of a P4 program

- Actions can then be applied in the *Ingress* part, using a match-action table. The controller configures entries of the table, with different actions to perform, based on input values of the packet headers. This match-action table system is at the heart of P4. The controller can, at any time, decide to change the configuration of the table, the entries, the actions to process, etc. This allows a dynamic configuration of the network. Then, the packets are placed in a queue (Traffic Manager). This part is currently non-programmable and this is one of the limitations of P4, which is not being able to program queues dynamically.
- Leaving this queue, packets are processed with actions and tables at the *Egress* part, in a similar way as in the Ingress.
- Finally, the *Deparser* reconstitutes the packets, adding the extracted (eventually modified) headers to the packet payload, and sends them to the network from the egress port.

Figure 32 summarizes the 4 main components with a skeleton of a P4 program.



```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
         out headers hdr,
         inout metadata meta,
         inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

Figure 32: Example of a simple P4 program

P4's advantages are manifold. It is a high-level, target-independent programming language which uses conventional constructs compiled to manage the resource and deal with the hardware. It introduces the programmability of devices to make them behave exactly as wanted, thus reducing the need and the risk of removing unused features in devices. Network devices used in different applications require different sets of protocols. Using P4, users can implement only the required protocols as per their application and remove protocols which are not required for their application. Thus, the available resources can be used effectively. It makes deployment of new protocols much simpler and consuming less time. Furthermore, as P4 programs can be written by the user, it helps to retain ownership of Intellectual Property. It eliminates the need of sharing new feature specifications with device vendors or their customers and, hence, improves the protection of confidential information and implementations.

P4 also has limitations. Since it is a domain specific programming language, it is difficult to extend it with new functionality. It is not easy to parse and deparse non-binary protocols, such as those that contain optional attributes or type-length-value (TLVs).

6.3.2 Low-latency performance and security in programmable devices using P4

In relation with the objectives of QoS for the MOSAICO project, the relevance of P4 in assisting the delivery of low-latency services is explored in very few papers. In this section, we review the related

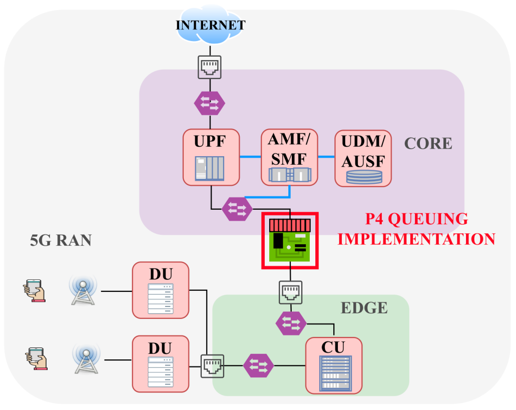


Figure 33: 5G edge to core architecture using P4-NetFPGA [154]

works using P4 for improving security and reducing latency in 5G networks. The application of P4 in this domain can be classified into 2 categories:

- Early detect and prevent malicious traffic [61, 153, 146]
- Archive low latency and improve QoS [168, 158, 100, 147, 199, 154]

The authors of [61] propose an in-network architecture for DDoS attack detection combining important traffic metrics of malicious traffic. These pertain to the number of flows and packet symmetry, maintained for protected subnets and utilized to identify anomalies. Appropriate alarms are triggered within time-based epochs and conveyed to external mitigation systems. The architecture has been assessed in P4-enabled SmartNICs in terms of detection accuracy and packet processing performance. As input to the accuracy experiments, the authors use real publicly available traffic traces. Results exhibit that this approach is applicable in typical enterprises and/or carrier environments, featuring packet rates of 1-2 Mpps for 10G links.

In [146], the authors leverage on the potential of P4 by applying P4 within a multi-layer edge scenario, proposing an architecture and its application on an SDN P4-enabled packet-over-optical node. Three specific multi-layer use cases covering dynamic stateful traffic engineering and cybersecurity are addressed through the P4-based solution. The experimental results using P4 in NetFPGA show that the P4-based DDoS detection introduces only an average latency of 5 μs in the range of 0-9.6Gbps. The latency increases up to 110 μs in the range 9.6-10Gbps upon quasi full-rate condition when NICs typically introduce significant packet loss.

The authors in [153] implement a new 5G Firewall that allows the detection, differentiation and selective blocking of 5G network traffic in the edge-to-core network segment of a 5G infrastructure developed using the P4 language. They use a hardware-accelerated framework based on FPGA. The authors also extended the framework in [154] to achieve network slicing between different 5G users in the edge-to-core network segment. The proposed framework provides hardware-isolation of the performance in terms of bandwidth, latency and packet loss of 5G network traffic. The experimental results are obtained from a real 5G infrastructure *in terms of end-to-end latency have been under 0.5ms for the highest priority queues and under 3ms for the lowest priority queues in an end-to-end communication.* Consequently, the framework fulfils the ambitious 5G end-to-end latency KPIs, defined as <1ms for critical traffic and and <10ms for normal traffic.

The architecture shown in Figure 33 is divided into three parts: 1) 5G RAN with antennas and 5G users connected to such antennas; 2) Edge with CUs to allow users mobility; and, 3) Core with the data centre hosting the core of the network. The proposed solution allows defining QoS network slicing over the 5G network traffic exchanged between Edge and Core.

The P4 parser extracts the 5G flow data that will be used as part of the definition of a network slice. A network slice is defined by the 6-tuple: 5G user source and destination IPs, 5G user source and destination ports, Differentiated Services Code Point (DSCP) and GTP Tunnel ID. Network slicing core provides a queuing discipline composed of 32 different queues which allow the definition of 32 types of QoS. It guarantees the complete isolation of the traffic in each queue. These queues have differentiated priorities, with queue 31 receiving the highest priority traffic and queue 0 the lowest. The queuing

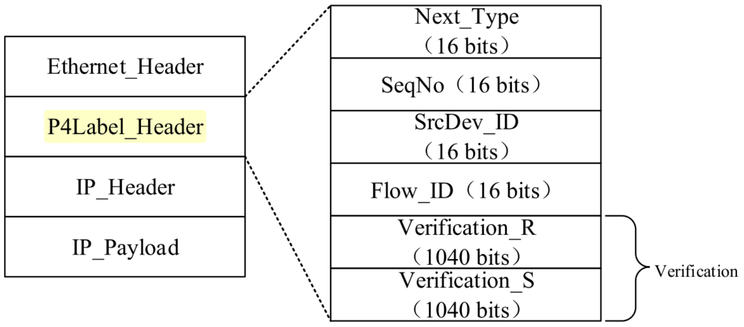


Figure 34: Specific composition of a P4Label header [199]

discipline is based on a priority-based algorithm where a queue with less priority will not be processed until the queues with higher priority are empty, enforcing a strict priority-based scheme.

The challenge of implementing, in P4, a fast-reroute algorithm compatible with stringent latency requirements is tackled in [168]. It addresses the case of port or link failures in P4 switches supporting low-latency services.

In [158], P4 is explored as a mean of providing exceptional low-latency control when the P4 switch is near a robot controlled by a remote program. This is achieved with a P4 program that reads the position information sent by the robot to its controller and that can craft control messages toward the robot in exceptional cases. Here, the edge location and line speed processing capacity are leveraged to assist low-latency service delivery. In these two papers, queueing latency is unaddressed.

Noticing the lack of modern AQM, such as CoDel and PIE, in deployed network equipment, the authors of [100] demonstrate and evaluate the implementation of CoDel using P4. Their objective is to fight bufferbloat for any type of Internet traffic rather than to distinguish the network service provided to low-latency traffic from the one provided to classical traffic.

Assuming that programmable traffic management at the data plane can lead to great benefits for QoS, the authors of [147] implement the PI2 Active Queue Management (AQM) scheme for reducing queuing delay using P4. They proved that implementing a modern AQM in P4 is not tricky and requires only basic bit manipulations at the data plane.

The authors of [199] address the following problems: limited match field that cannot meet the precise control of the network services, lack of an effective data source verification mechanism, and terminal devices need of public key certificates that complicate the system and have high maintenance costs. For this, they propose an SDN packet forwarding control mechanism based on P4. The authors add a new P4Label protocol header to the packets to implement three basic packet forwarding control functions: 1) flow identification function, 2) identity function, and 3) verification function.

The flow identification function refers to designing the `Flow_ID` field value in the header according to specific information of important objects such as users and network services, and differentiating various data flows according to this field value. Through this function, P4Label is used as the matching identifier that the forwarding devices can recognize so that the network packet forwarding behavior can be defined based on the `Flow_ID` field. The identity function refers to binding the P4Label header to the source device identity of the sender and identifying the packet source device through the `SrcDev_ID` field. Through this function, the IBS algorithm can be used to generate public and private keys of the source device and simultaneously generate packet signature information. The network forwarding devices do not need public key certificate or certificate authority to obtain the sender's public key for data verification, which reduces system maintenance costs and complexity. The verification function refers to storing signature information in the Verification field in the P4Label header. Through this function, the forwarding device in the network can verify the Verification field, implement source verification and ensure the integrity of the packet, and prevent users from tampering legitimate data packets. Performance analysis shows that P4Label takes an average of 0.99ms (or 1.12ms) for each IP packet that is 74 bytes (or respectively 342 bytes) in size.

7 Conclusion

The survey for classifying low-latency applications showed that these applications are very different in terms of latency requirements, data volume exchange (from the server to clients or vice-versa), the used transport protocol, etc. It is therefore challenging to extract common features for defining LL applications. Regarding the various causes of latency and focusing more specifically on network-based causes, such as those we address in the MOSAICO project, we abstracted the LL applications into classes. Three categories have been identified, two for low-latency applications differentiated by their capability to dynamically adapt to network conditions, one for extreme low-latency services. In the next steps of the project, we will more deeply investigate solutions for applications having a latency requirements lower than 100ms, delivered over TCP and over RTP/UDP, and more finely analyse the concerned applications such as cloud gaming, cloud robotics or Realtime VR streaming.

The survey on network solutions to improve latency allowed us to detect that the deployed active queue management solution and the associated congestion control algorithms can have a huge impact on the latency and on the protocol reactivity. Based on this analysis, we decided to further investigate the L4S approach, which looks like the most promising for our objectives.

Regarding the orchestration, our survey clearly highlights the huge related work on orchestration solutions for network services, but it also shows that that there are currently only few relevant works on microservices orchestration which appears as an emerging topic. Being one objective of the MOSAICO project, it allows us to define and propose innovative solutions in this field.

The security of low-latency applications is also an emerging topic, not yet addressed by many published papers or existing projects. Our survey enables us to identify few works related to attacks using hacked ACK, hacked ECN or low-rate DoS, but there is not much research on attack for the L4S architecture (e.g., using specific burst traffic) which could perturb the on-time delivery of low-latency services. This will be one of the security issues we will further investigate in the project.

Finally, the analysis of different candidate technologies leads us to promote the usage of P4 for in-network data processing that can be used by rather simple modules not requiring difficult processing tasks but achieving line rate speeds, OpenNetVM for more demanding modules where processing can be done in the user space of machines and adopting NFV concepts.

Consequently, the next steps of the project will be to design, develop and evaluate the performance and security of novel LL modules of the project, and start the orchestration part for the deployment of the appropriate modules in the right location and the right technology.

References

[1] *3GPP TR 36.777 Study on Enhanced LTE Support for Aerial Vehicles*. Tech. rep. 2018. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3231.

[2] Raz Abramov and Amir Herzberg. “TCP Ack storm DoS attacks”. In: *Computers & Security* 33 (2013). Future Challenges in Security and Privacy for Academia and Industry, pp. 12–27. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2012.09.005. URL: http://www.sciencedirect.com/science/article/pii/S0167404812001411.

[3] R. Adams. “Active Queue Management: A Survey”. In: *IEEE Communications Surveys Tutorials* 15.3 (2013), pp. 1425–1476. DOI: 10.1109/SURV.2012.082212.00018.

[4] F. Adamsky et al. “Security Analysis of the Micro Transport Protocol with a Misbehaving Receiver”. In: *2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. 2012, pp. 143–150. DOI: 10.1109/CyberC.2012.31.

[5] Boussad Addad, Saïd Amari, and Jean-Jacques Lesage. “Analytic Calculus of Response Time in Networked Automation Systems”. In: *IEEE Transactions on Automation Science and Engineering* 7.4 (Apr. 2010), pp. 858–869. DOI: 10.1109/TASE.2010.2047499. URL: https://hal.archives-ouvertes.fr/hal-00514886.

[6] B. Addis et al. “Virtual network functions placement and routing optimization”. In: *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*. 2015, pp. 171–177. DOI: 10.1109/CloudNet.2015.7335301.

[7] Bernardetta Addis, Giuliana Carello, and Meihui Gao. “On a virtual network functions placement and routing problem: Some properties and a comparison of two formulations”. In: *Networks* 75 (Nov. 2019). DOI: 10.1002/net.21915.

[8] Bernardetta Addis, Meihui Gao, and Giuliana Carello. “On the complexity of a Virtual Network Function Placement and Routing problem”. In: *Electronic Notes in Discrete Mathematics* 69 (2018). Joint EURO/ALIO International Conference 2018 on Applied Combinatorial Optimization (EURO/ALIO 2018), pp. 197–204. ISSN: 1571-0653. DOI: https://doi.org/10.1016/j.endm.2018.07.026. URL: http://www.sciencedirect.com/science/article/pii/S1571065318301707.

[9] Adnan Aijaz et al. “Realizing the Tactile Internet: Haptic Communications over Next Generation 5G Cellular Networks”. In: *IEEE Wirel. Commun.* 24.2 (2017), pp. 82–89. DOI: 10.1109/MWC.2016.1500157RP. URL: https://doi.org/10.1109/MWC.2016.1500157RP.

[10] O. Ait-Hellal and E. Altman. “Analysis of TCP Vegas and TCP Reno”. In: *Proceedings of ICC’97 - International Conference on Communications*. Vol. 1. 1997, 495–499 vol.1. DOI: 10.1109/ICC.1997.605357.

[11] M. Alam et al. “Orchestration of Microservices for IoT Using Docker and Edge Computing”. In: *IEEE Communications Magazine* 56.9 (2018), pp. 118–123. DOI: 10.1109/MCOM.2018.1701233.

[12] O. Albisser et al. “DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM”. In: *Proc. Linux Netdev 0x13* (Mar. 2019). URL: https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM.

[13] O. Alhussein et al. “Joint VNF Placement and Multicast Traffic Routing in 5G Core Networks”. In: *2018 IEEE Global Communications Conference (GLOBECOM)*. 2018, pp. 1–6. DOI: 10.1109/GLOCOM.2018.8648029.

[14] Zaid Allybokus et al. “Virtual Function Placement for Service Chaining with Partial Orders and Anti-Affinity Rules”. In: *Networks* 71 (May 2017). DOI: 10.1002/net.21768.

[15] Sotiris Avgousti et al. “Medical telerobotic systems: current status and future trends”. In: *BioMedical Engineering OnLine* 15.1 (Aug. 2016). DOI: 10.1186/s12938-016-0217-7. URL: https://doi.org/10.1186.

[16] Fred Baker and Gorry Fairhurst. *IETF Recommendations Regarding Active Queue Management*. RFC 7567. July 2015. DOI: 10.17487/RFC7567. URL: https://rfc-editor.org/rfc/rfc7567.txt.

[17] F. Bari et al. “Orchestrating Virtualized Network Functions”. In: *IEEE Transactions on Network and Service Management* 13.4 (2016), pp. 725–739. DOI: 10.1109/TNSM.2016.2569020.

[18] M. F. Bari et al. “On orchestrating virtual network functions”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. 2015, pp. 50–56. DOI: 10.1109/CNSM.2015.7367338.

[19] Carlos Barriquello et al. “Fundamentals of Wireless Communication Link Design for Networked Robotics”. In: Jan. 2018. ISBN: 978-953-51-3722-1. DOI: 10.5772/intechopen.69873.

[20] Stephen Bensley et al. *Data Center TCP (DCTCP): TCP Congestion Control for Data Centers*. RFC 8257. Oct. 2017. DOI: 10.17487/RFC8257. URL: https://rfc-editor.org/rfc/rfc8257.txt.

[21] Abdelhak Bentaleb et al. “Bandwidth prediction in low-latency chunked streaming”. In: *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video, NOSSDAV 2019, Amherst, MA, USA, June 21, 2019*. Ed. by Ali C. Begen, Thomas Schierl, and Sejin Oh. ACM, 2019, pp. 7–13. DOI: 10.1145/3304112.3325611. URL: https://doi.org/10.1145/3304112.3325611.

[22] Deval Bhamare et al. “Multi-objective scheduling of micro-services for optimal service function chains”. In: *2017 IEEE International Conference on Communications (ICC)*. 2017, pp. 1–6. DOI: 10.1109/ICC.2017.7996729.

[23] Monowar H. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita. “Surveying Port Scans and Their Detection Methodologies”. In: *The Computer Journal* 54.10 (Apr. 2011), pp. 1565–1581. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxr035. eprint: https://academic.oup.com/comjnl/article-pdf/54/10/1565/1775166/bxr035.pdf. URL: https://doi.org/10.1093/comjnl/bxr035.

[24] David L. Black. *Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation*. RFC 8311. Jan. 2018. DOI: 10.17487/RFC8311. URL: https://rfc-editor.org/rfc/rfc8311.txt.

[25] David L. Black et al. *An Architecture for Differentiated Services*. RFC 2475. Dec. 1998. DOI: 10.17487/RFC2475. URL: https://rfc-editor.org/rfc/rfc2475.txt.

[26] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control*. RFC 5681. Sept. 2009. DOI: 10.17487/RFC5681. URL: https://rfc-editor.org/rfc/rfc5681.txt.

[27] Marlene Bohmer et al. “Latency-aware and -predictable Communication with Open Protocol Stacks for Remote Drone Control”. In: *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)* (May 2020). DOI: 10.1109/dcoss49796.2020.00055. URL: http://dx.doi.org/10.1109/DCOSS49796.2020.00055.

[28] Marlene Böhmer et al. “Latency-aware and -predictable Communication with Open Protocol Stacks for Remote Drone Control”. In: *16th International Conference on Distributed Computing in Sensor Systems, DCOSS 2020, Marina del Rey, CA, USA, May 25-27, 2020*. IEEE, 2020, pp. 304–311. DOI: 10.1109/DCOSS49796.2020.00055. URL: https://doi.org/10.1109/DCOSS49796.2020.00055.

[29] R. B. Bohn et al. “NIST Cloud Computing Reference Architecture”. In: *2011 IEEE World Congress on Services*. 2011, pp. 594–596. DOI: 10.1109/SERVICES.2011.105.

[30] Dejene Boru Oljira et al. “Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture”. In: *SIGCOMM Comput. Commun. Rev.* 50.2 (May 2020), pp. 37–44. ISSN: 0146-4833. DOI: 10.1145/3402413.3402419. URL: https://doi-org.proxy.utt.fr/10.1145/3402413.3402419.

[31] Pat Bosshart et al. “P4: Programming Protocol-Independent Packet Processors”. In: *SIGCOMM Comput. Commun. Rev.* 44.3 (July 2014), pp. 87–95. ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. URL: https://doi.org/10.1145/2656877.2656890.

[32] Lawrence S. Brakmo, Sean W. O’Malley, and Larry L. Peterson. “TCP Vegas: New Techniques for Congestion Detection and Avoidance”. In: *Proceedings of the ACM SIGCOMM ’94 Conference on Communications Architectures, Protocols and Applications, London, UK, August 31 - September 2, 1994*. Ed. by Jon Crowcroft. ACM, 1994, pp. 24–35. DOI: 10.1145/190314.190317. URL: https://doi.org/10.1145/190314.190317.

[33] Anat Bremler-Barr, Yotam Harchol, and David Hay. "OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. SIGCOMM '16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 511–524. ISBN: 9781450341936. DOI: 10.1145/2934872.2934875. URL: https://doi.org/10.1145/2934872.2934875.

[34] B. Briscoe, M. Kuehlewind, and R. Scheffenegger. *More Accurate ECN Feedback in TCP, draft-ietf-tcpm-accurate-ecn-11 (work in progress)*. [Online; last accessed 01/2021]. 2020. URL: https://tools.ietf.org/html/draft-ietf-tcpm-accurate-ecn-13.

[35] B. Briscoe et al. "Implementing the 'Prague Requirements'for Low Latency Low Loss Scalable Throughput (L4S)". In: *Netdev0x13* (2019).

[36] B. Briscoe et al. "Reducing Internet Latency: A Survey of Techniques and Their Merits". In: *IEEE Communications Surveys Tutorials* 18.3 (2016), pp. 2149–2196. DOI: 10.1109/COMST.2014.2375213.

[37] Bob Briscoe and Greg White. *Queue Protection to Preserve Low Latency*. Internet-Draft draft-briscoe-docsis-q-protection-00. Work in Progress. Internet Engineering Task Force, July 2019. 24 pp. URL: https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-00.

[38] Bob Briscoe et al. *Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture*. Internet-Draft draft-ietf-tsvwg-l4s-arch-08. Work in Progress. Internet Engineering Task Force, Nov. 2020. 41 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-08.

[39] Peter Brucker. *Scheduling Algorithms*. Vol. 47. Jan. 2004. DOI: 10.2307/3010416.

[40] *BT-1359, relative timing of sound and vision for broadcasting*. [Online; last accessed 01/2021]. 1998. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1359-0-199802-S!!PDF-E.pdf.

[41] Chao Bu et al. "Enabling Adaptive Routing Service Customization via the Integration of SDN and NFV". In: *Journal of Network and Computer Applications* 93 (June 2017). DOI: 10.1016/j.jnca.2017.05.010.

[42] Mihai Budiu and Chris Dodd. "The P416 Programming Language". In: *SIGOPS Oper. Syst. Rev.* 51.1 (Sept. 2017), pp. 5–14. ISSN: 0163-5980. DOI: 10.1145/3139645.3139648. URL: https://doi.org/10.1145/3139645.3139648.

[43] Jaroslaw Byrka et al. "An Improved LP-based Approximation for Steiner Tree". In: *Proc. 42nd STOC* (Dec. 2010). DOI: 10.1145/1806689.1806769.

[44] Wei Cai et al. "A Survey on Cloud Gaming: Future of Computer Games". In: *IEEE Access* 4 (2016), pp. 7605–7620. DOI: 10.1109/ACCESS.2016.2590500. URL: https://doi.org/10.1109/ACCESS.2016.2590500.

[45] Neal Cardwell et al. "BBR: Congestion-Based Congestion Control". In: *ACM Queue* 14.5 (2016), pp. 20–53. DOI: 10.1145/3012426.3022184. URL: http://doi.acm.org/10.1145/3012426.3022184.

[46] Marc Carrascosa and Boris Bellalta. "Cloud-gaming: Analysis of Google Stadia traffic". In: *CoRR* abs/2009.09786 (2020). arXiv: 2009.09786. URL: https://arxiv.org/abs/2009.09786.

[47] Martín Casado et al. "Virtualizing the network forwarding plane". In: (Jan. 2010), p. 8. DOI: 10.1145/1921151.1921162.

[48] Kuan-Ta Chen, Polly Huang, and Chin-Laung Lei. "Effect of Network Quality on Player Departure Behavior in Online Games". In: *IEEE Trans. Parallel Distributed Syst.* 20.5 (2009), pp. 593–606. DOI: 10.1109/TPDS.2008.148. URL: https://doi.org/10.1109/TPDS.2008.148.

[49] Kuan-Ta Chen et al. "Measuring the latency of cloud gaming systems". In: *Proceedings of the 19th International Conference on Multimedia 2011, Scottsdale, AZ, USA, November 28 - December 1, 2011*. Ed. by K. Selçuk Candan et al. ACM, 2011, pp. 1269–1272. DOI: 10.1145/2072298.2071991. URL: https://doi.org/10.1145/2072298.2071991.

[50] W. Chen et al. "A Study of Robotic Cooperation in Cloud Robotics: Architecture and Challenges". In: *IEEE Access* 6 (2018), pp. 36662–36682. DOI: 10.1109/ACCESS.2018.2852295.

[51] Margaret Chiosi et al. *Network Functions Virtualisation (NFV) Network Operator Perspectives on Industry Progress*. Oct. 2013. DOI: 10.13140/RG.2.1.4110.2883.

[52] S. R. Chowdhury et al. "Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges". In: *IEEE Network* 33.3 (2019), pp. 168–176. DOI: 10.1109/MNET.2019.1800082.

[53] S. R. Chowdhury et al. "Re-Architecting NFV Ecosystem with Microservices: State of the Art and Research Challenges". In: *IEEE Network* 33.3 (2019), pp. 168–176.

[54] Shihabur Chowdhury et al. "µNF: A Disaggregated Packet Processing Architecture". In: June 2019. DOI: 10.1109/NETSOFT.2019.8806657.

[55] Shihabur Rahman Chowdhury et al. "µNF: A Disaggregated Packet Processing Architecture". In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. 2019, pp. 342–350. DOI: 10.1109/NETSOFT.2019.8806657.

[56] Maxim Claeys et al. "Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client". In: *IEEE Commun. Lett.* 18.4 (2014), pp. 716–719. DOI: 10.1109/LCOMM.2014.020414.132649. URL: https://doi.org/10.1109/LCOMM.2014.020414.132649.

[57] Mark Claypool et al. "Thin to win? Network performance analysis of the OnLive thin client game system". In: *11th Annual Workshop on Network and Systems Support for Games, NetGames 2012, Venice, Italy, November 22-23, 2012*. IEEE, 2012, pp. 1–6. DOI: 10.1109/NetGames.2012.6404013. URL: https://doi.org/10.1109/NetGames.2012.6404013.

[58] R. Cohen et al. "Near optimal placement of virtual network functions". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. 2015, pp. 1346–1354. DOI: 10.1109/INFOCOM.2015.7218511.

[59] Koen De Schepper et al. "PI^2: A Linearized AQM for Both Classic and Scalable TCP". In: *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '16. Irvine, California, USA: Association for Computing Machinery, 2016, pp. 105–119. ISBN: 9781450342926. DOI: 10.1145/2999572.2999578. URL: https://doi.org/10.1145/2999572.2999578.

[60] Adam C. Desveaux and Valerie S. Courtemanche. *The Effects of Latency in Commercial Cloud Video Gaming Services*. Tech. rep. 100 Institute Road, Worcester MA 01609-2280 USA: Worcester Polytechnic Institute, Mar. 2020.

[61] Marinos Dimolianis, Adam Pavlidis, and Vasilis Maglaris. "A Multi-Feature DDoS Detection Schema on P4 Network Hardware". In: *Workshop on Flexible Network Data Plane Processing - NETPROC@ICIN*. Mis 100815. IEEE, 2020, pp. 1–6. ISBN: 9781728151274. DOI: 10.1109/ICIN48450.2020.9059327.

[62] Nicola Dragoni et al. "Microservices: yesterday, today, and tomorrow". In: *Present and Ulterior Software Engineering*. Ed. by Manuel Mazzara and Bertrand Meyer. Springer, Sept. 2017. URL: https://hal.inria.fr/hal-01631455.

[63] Qiang Duan, Nirwan Ansari, and Mehmet Toy. "Software-defined network virtualization: An architectural framework for integrating SDN and NFV for service provisioning in future networks". In: *IEEE Network* 30 (Sept. 2016), pp. 10–16. DOI: 10.1109/MNET.2016.7579021.

[64] D. Ely et al. "Robust congestion signaling". In: *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*. 2001, pp. 332–341. DOI: 10.1109/ICNP.2001.992914.

[65] ETSI. "2014e. Network Functions Virtualisation (NFV); management and orchestration; report on architectural options. technical report". In: (). Accessed February 2019. [Online]. URL: http://www.etsi.org/deliver/etsi%20gs/NFV-IFA/001%20099/%20009/01.01.01%2060/gs%20NFV-IFA009v010101p.pdf.

[66] S. Euler et al. "Mobility Support for Cellular Connected Unmanned Aerial Vehicles: Performance and Analysis". In: *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. 2019, pp. 1–6. DOI: 10.1109/WCNC.2019.8885820.

[67] Davide Falanga, Suseong Kim, and Davide Scaramuzza. "How Fast Is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid". In: *IEEE Robotics Autom. Lett.* 4.2 (2019), pp. 1884–1891. DOI: 10.1109/LRA.2019.2898117. URL: https://doi.org/10.1109/LRA.2019.2898117.

[68] Wu-chang Feng, Dilip Kandlur, and Debanjan Saha. "The BLUE active queue management algorithms". In: *Networking, IEEE/ACM Transactions on* 10 (Sept. 2002), pp. 513–528. DOI: 10.1109/TNET.2002.801399.

[69] Gerhard P. Fettweis. "The tactile internet: Applications and challenges". In: *Ieee Vehicular Technology Magazine* 9.1 (2014), pp. 64–70. DOI: 10.1109/MVT.2013.2295069. URL: http://ieeexplore.ieee.org/document/6755599/.

[70] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/faq.htm.

[71] Sally Floyd and Thomas R. Henderson. "The NewReno Modification to TCP's Fast Recovery Algorithm". In: *RFC* 2582 (1999), pp. 1–12. DOI: 10.17487/RFC2582. URL: https://doi.org/10.17487/RFC2582.

[72] Sally Floyd and Van Jacobson. "Random early detection gateways for congestion avoidance". In: *IEEE/ACM Trans. Netw.* 1.4 (1993), pp. 397–413. DOI: 10.1109/90.251892. URL: https://doi.org/10.1109/90.251892.

[73] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP.* RFC 3168. Sept. 2001. DOI: 10.17487/RFC3168. URL: https://rfc-editor.org/rfc/rfc3168.txt.

[74] Linux Sofware Foundation. *Data Plane Development Kit.* URL: https://www.dpdk.org/ (visited on 05/07/2020).

[75] *G-107, The E-model: a computational model for use in transmission planning.* [Online; last accessed 01/2021]. 2015. URL: https://www.itu.int/rec/T-REC-G.107-201506-I/en.

[76] *G-114, One-way transmission time.* [Online; last accessed 01/2021]. 2003. URL: https://www.itu.int/rec/T-REC-G.114-200305-I/en.

[77] Chang Ge et al. "QoE-Assured 4K HTTP Live Streaming via Transient Segment Holding at Mobile Edge". In: *IEEE J. Sel. Areas Commun.* 36.8 (2018), pp. 1816–1830. DOI: 10.1109/JSAC.2018.2845000. URL: https://doi.org/10.1109/JSAC.2018.2845000.

[78] Aaron Gember-Jacobson and Aditya Akella. "Improving the Safety, Scalability, and Efficiency of Network Function State Transfers". In: Aug. 2015, pp. 43–48. DOI: 10.1145/2785989.2785997.

[79] R. Gouareb, V. Friderikos, and A. H. Aghvami. "Delay Sensitive Virtual Network Function Placement and Routing". In: *2018 25th International Conference on Telecommunications (ICT).* 2018, pp. 394–398. DOI: 10.1109/ICT.2018.8464883.

[80] M. Guirguis et al. "Reduction of quality (RoQ) attacks on Internet end-systems". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 2. 2005, 1362–1372 vol. 2. DOI: 10.1109/INFCOM.2005.1498361.

[81] Abhishek Gupta et al. "On service-chaining strategies using Virtual Network Functions in operator networks". In: *Computer Networks* 133 (2018), pp. 1–16. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2018.01.028. URL: http://www.sciencedirect.com/science/article/pii/S1389128618300379.

[82] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant". In: *ACM SIGOPS Oper. Syst. Rev.* 42.5 (2008), pp. 64–74. DOI: 10.1145/1400097.1400105. URL: http://doi.acm.org/10.1145/1400097.1400105.

[83] Hassan Hawilo, Manar Jammal, and Abdallah Shami. "Exploring Microservices as the Architecture of Choice for Network Function Virtualization Platforms". In: *IEEE Network* 33.2 (2019), pp. 202–210. DOI: 10.1109/MNET.2019.1800023.

[84] Toke Høiland-Jørgensen et al. *The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm.* RFC 8290. Jan. 2018. DOI: 10.17487/RFC8290. URL: https://rfc-editor.org/rfc/rfc8290.txt.

[85] C. V. Hollot et al. "On designing improved controllers for AQM routers supporting TCP flows". In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213).* Vol. 3. 2001, 1726–1734 vol.3. DOI: 10.1109/INFCOM.2001.916670.

[86] Jeroen van der Hooft et al. “An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments”. In: *J. Netw. Syst. Manag.* 26.1 (2018), pp. 51–78. DOI: 10.1007/s10922-017-9407-2. URL: https://doi.org/10.1007/s10922-017-9407-2.

[87] N. Hoque et al. “Network attacks: Taxonomy, tools and systems”. In: *Journal of Network and Computer Applications* 40 (2014), pp. 307–324. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2013.08.001.

[88] H. Huang et al. “Service Chaining for Hybrid Network Function”. In: *IEEE Transactions on Cloud Computing* 7.4 (2019), pp. 1082–1094. DOI: 10.1109/TCC.2017.2721401.

[89] Narjes Tahghigh Jahromi et al. “An NFV and microservice based architecture for on-the-fly component provisioning in content delivery networks”. In: *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2018, pp. 1–7. DOI: 10.1109/CCNC.2018.8319227.

[90] Narjes Tahghigh Jahromi et al. “An NFV and microservice based architecture for on-the-fly component provisioning in content delivery networks”. In: *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 2018, pp. 1–7. DOI: 10.1109/CCNC.2018.8319227.

[91] Ewa Czeslawa JANCZUKOWICZ. “QoS management for WebRTC : loose coupling strategies”. Theses. Ecole nationale supérieure Mines-Télécom Atlantique, Mar. 2017. URL: https://tel.archives-ouvertes.fr/tel-01781632.

[92] Sofiene Jelassi et al. “Quality of Experience of VoIP Service: A Survey of Assessment Approaches and Open Issues”. In: *IEEE Commun. Surv. Tutorials* 14.2 (2012), pp. 491–513. DOI: 10.1109/SURV.2011.120811.00063. URL: https://doi.org/10.1109/SURV.2011.120811.00063.

[93] B. Kehoe et al. “A Survey of Research on Cloud Robotics and Automation”. In: *IEEE Transactions on Automation Science and Engineering* 12.2 (2015), pp. 398–409. DOI: 10.1109/TASE.2014.2376492.

[94] Ben Kehoe et al. “A Survey of Research on Cloud Robotics and Automation”. In: *IEEE Trans Autom. Sci. Eng.* 12.2 (2015), pp. 398–409. DOI: 10.1109/TASE.2014.2376492. URL: https://doi.org/10.1109/TASE.2014.2376492.

[95] N. Kiji et al. “Virtual Network Function Placement and Routing Model for Multicast Service Chaining Based on Merging Multiple Service Paths”. In: *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)*. 2019, pp. 1–6. DOI: 10.1109/HPSR.2019.8807998.

[96] Eddie Kohler et al. “The Click Modular Router”. In: *ACM Trans. Comput. Syst.* 18.3 (Aug. 2000), pp. 263–297. ISSN: 0734-2071. DOI: 10.1145/354871.354874. URL: https://doi.org/10.1145/354871.354874.

[97] Nupur Kothari et al. “Finding Protocol Manipulation Attacks”. In: *SIGCOMM Comput. Commun. Rev.* 41.4 (Aug. 2011), pp. 26–37. ISSN: 0146-4833. DOI: 10.1145/2043164.2018440. URL: https://doi.org/10.1145/2043164.2018440.

[98] Sameer Kulkarni et al. “NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains”. In: Aug. 2017, pp. 71–84. DOI: 10.1145/3098822.3098828.

[99] Sameer G Kulkarni et al. “REINFORCE: Achieving Efficient Failure Resiliency for Network Function Virtualization Based Services”. In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT ’18. Heraklion, Greece: Association for Computing Machinery, 2018, pp. 41–53. ISBN: 9781450360807. DOI: 10.1145/3281411.3281441. URL: https://doi.org/10.1145/3281411.3281441.

[100] Ralf Kundel et al. “P4-CoDel: Active Queue Management in Programmable Data Planes”. In: *IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2018, Verona, Italy, November 27-29, 2018*. IEEE, 2018, pp. 1–4. DOI: 10.1109/NFV-SDN.2018.8725736. URL: https://doi.org/10.1109/NFV-SDN.2018.8725736.

[101] Aleksandar Kuzmanovic and Edward W. Knightly. “Low-Rate TCP-Targeted Denial of Service Attacks: The Shrew vs. the Mice and Elephants”. In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM ’03. Karlsruhe, Germany: Association for Computing Machinery, 2003, pp. 75–86. ISBN: 1581137354. DOI: 10.1145/863955.863966. URL: https://doi.org/10.1145/863955.863966.

[102] A. Laraba et al. “Defeating Protocol Abuse with P4: Application to Explicit Congestion Notification”. In: *2020 IFIP Networking Conference (Networking)*. 2020, pp. 431–439.

[103] G. Larysa, S. Mariia, and S. Svitlana. "Method for resource allocation of virtualized network functions in hybrid environment". In: *2016 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. 2016, pp. 1–5. DOI: 10.1109/BlackSeaCom.2016.7901546.

[104] Giwon Lee et al. "Optimal Flow Distribution in Service Function Chaining". In: *The 10th International Conference on Future Internet*. CFI '15. Seoul, Republic of Korea: Association for Computing Machinery, 2015, pp. 17–20. ISBN: 9781450335645. DOI: 10.1145/2775088.2775103. URL: https://doi.org/10.1145/2775088.2775103.

[105] Kyungmin Lee et al. "Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming". In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2015, Florence, Italy, May 19-22, 2015*. Ed. by Gaetano Borriello et al. ACM, 2015, pp. 151–165. DOI: 10.1145/2742647.2742656. URL: https://doi.org/10.1145/2742647.2742656.

[106] Yeng-Ting Lee et al. "Are all games equally cloud-gaming-friendly? An electromyographic approach". In: *11th Annual Workshop on Network and Systems Support for Games, NetGames 2012, Venice, Italy, November 22-23, 2012*. IEEE, 2012, pp. 1–6. DOI: 10.1109/NetGames.2012.6404025. URL: https://doi.org/10.1109/NetGames.2012.6404025.

[107] Youngki Lee et al. "Measurement and Estimation of Network QoS Among Peer Xbox 360 Game Players". In: *Passive and Active Network Measurement, 9th International Conference, PAM 2008, Cleveland, OH, USA, April 29-30, 2008. Proceedings*. Ed. by Mark Claypool and Steve Uhlig. Vol. 4979. Lecture Notes in Computer Science. Springer, 2008, pp. 41–50. DOI: 10.1007/978-3-540-79232-1_5. URL: https://doi.org/10.1007/978-3-540-79232-1%5C_5.

[108] J. Li et al. "Delay-Aware VNF Scheduling: A Reinforcement Learning Approach With Variable Action Set". In: *IEEE Transactions on Cognitive Communications and Networking* 7.1 (2021), pp. 304–318. DOI: 10.1109/TCCN.2020.2988908.

[109] J. Li et al. "Reinforcement Learning Based VNF Scheduling with End-to-End Delay Guarantee". In: *2019 IEEE/CIC International Conference on Communications in China (ICCC)*. 2019, pp. 572–577. DOI: 10.1109/ICCChina.2019.8855889.

[110] Taixin Li, Zhou Huachun, and Hongbin Luo. "A New Method for Providing Network Services: Service Function Chain". In: *Optical Switching and Networking* 26 (Sept. 2015). DOI: 10.1016/j.osn.2015.09.005.

[111] Xiaozhou Li and Michael Freedman. "Scaling IP multicast on datacenter topologies". In: Dec. 2013, pp. 61–72. DOI: 10.1145/2535372.2535380.

[112] Z. Li and Y. Yang. "Placement of Virtual Network Functions in Hybrid Data Center Networks". In: *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*. 2017, pp. 73–79. DOI: 10.1109/HOTI.2017.15.

[113] J. Liu et al. "On Dynamic Service Function Chain Deployment and Readjustment". In: *IEEE Transactions on Network and Service Management* 14.3 (2017), pp. 543–553. DOI: 10.1109/TNSM.2017.2711610.

[114] D. M. Lofaro, A. Asokan, and E. M. Roderick. "Feasibility of cloud enabled humanoid robots: Development of low latency geographically adjacent real-time cloud control". In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 519–526. DOI: 10.1109/HUMANOIDS.2015.7363582.

[115] *Low-latency design considerations for video-enabled drones*. [Online; last accessed 01/2021]. 2016. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1359-0-199802-S!!PDF-E.pdf.

[116] F. Lucrezia et al. "Introducing network-aware scheduling capabilities in OpenStack". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. 2015, pp. 1–5. DOI: 10.1109/NETSOFT.2015.7116155.

[117] M. C. Luizelli et al. "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 98–106. DOI: 10.1109/INM.2015.7140281.

[118] Marcelo Caggiani Luizelli et al. “A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining”. In: *Computer Communications* 102 (2017), pp. 67–77. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2016.11.002. URL: http://www.sciencedirect.com/science/article/pii/S0140366416305485.

[119] Xiaohui Luo, Fengyuan Ren, and Tong Zhang. “High Performance Userspace Networking for Containerized Microservices”. In: *Service-Oriented Computing*. Ed. by Claus Pahl et al. Cham: Springer International Publishing, 2018, pp. 57–72. ISBN: 978-3-030-03596-9. DOI: 10.1007/978-3-030-03596-9_4.

[120] A. T. Maereg et al. “Wearable Vibrotactile Haptic Device for Stiffness Discrimination during Virtual Interactions”. In: *Frontiers Robotics AI* 4 (2017), p. 42.

[121] Marc Manzano et al. “An empirical study of Cloud Gaming”. In: *11th Annual Workshop on Network and Systems Support for Games, NetGames 2012, Venice, Italy, November 22-23, 2012*. IEEE, 2012, pp. 1–2. DOI: 10.1109/NetGames.2012.6404021. URL: https://doi.org/10.1109/NetGames.2012.6404021.

[122] Marc Manzano et al. “Dissecting the protocol and network traffic of the OnLive cloud gaming platform”. In: *Multim. Syst.* 20.5 (2014), pp. 451–470. DOI: 10.1007/s00530-014-0370-4. URL: https://doi.org/10.1007/s00530-014-0370-4.

[123] Matthew Marsden, Mike Hazas, and Matthew Broadbent. “From One Edge to the Other: Exploring Gaming’s Rising Presence on the Network”. In: *ICT4S 2020: 7th International Conference on ICT for Sustainability, Bristol, United Kingdom, June 21-27, 2020*. Ed. by Ruzanna Chitchyan et al. ACM, 2020, pp. 247–254. DOI: 10.1145/3401335.3401366. URL: https://doi.org/10.1145/3401335.3401366.

[124] R. Mathew and V. Katkar. “Survey of Low Rate DoS Attack Detection Mechanisms”. In: *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*. ICWET ’11. Mumbai, Maharashtra, India: Association for Computing Machinery, 2011, pp. 955–958. ISBN: 9781450304498. DOI: 10.1145/1980022.1980227.

[125] Matt Mathis and Bob Briscoe. *Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements*. RFC 7713. Dec. 2015. DOI: 10.17487/RFC7713. URL: https://rfc-editor.org/rfc/rfc7713.txt.

[126] S. Mehraghdam, M. Keller, and H. Karl. “Specifying and placing chains of virtual network functions”. In: *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. 2014, pp. 7–13. DOI: 10.1109/CloudNet.2014.6968961.

[127] B Meindl and Matthias Templ. “Analysis of commercial and free and open source solvers for linear optimization problems”. In: (Aug. 2013).

[128] Zili Meng et al. “MicroNF: An Efficient Framework for Enabling Modularized Service Chains in NFV”. In: *IEEE Journal on Selected Areas in Communications* 37.8 (2019), pp. 1851–1865. DOI: 10.1109/JSAC.2019.2927069.

[129] Anu Mercian et al. “Software Defined Optical Networks (SDONs): A Comprehensive Survey”. In: *IEEE Communications Surveys & Tutorials* 18 (Nov. 2015). DOI: 10.1109/COMST.2016.2586999.

[130] R. Mijumbi et al. “Design and evaluation of algorithms for mapping and scheduling of virtual network functions”. In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. 2015, pp. 1–9. DOI: 10.1109/NETSOFT.2015.7116120.

[131] H. Moens and F. D. Turck. “VNF-P: A model for efficient placement of virtualized network functions”. In: *10th International Conference on Network and Service Management (CNSM) and Workshop*. 2014, pp. 418–423. DOI: 10.1109/CNSM.2014.7014205.

[132] T. Mononen and J. Mattila. “A low-cost cloud-extended sensor network for supervisory control”. In: *2017 IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. 2017, pp. 502–507. DOI: 10.1109/ICCIS.2017.8274827.

[133] Siva D. Muruganathan et al. “An Overview of 3GPP Release-15 Study on Enhanced LTE Support for Connected Drones”. In: *CoRR* abs/1805.00826 (2018). arXiv: 1805.00826. URL: http://arxiv.org/abs/1805.00826.

[134] Szilveszter Nádas et al. “A Congestion Control Independent L4S Scheduler”. In: *Proceedings of the Applied Networking Research Workshop*. ANRW ’20. Virtual Event, Spain: Association for Computing Machinery, 2020, pp. 45–51. ISBN: 9781450380393. DOI: 10.1145/3404868.3406669. URL: https://doi.org/10.1145/3404868.3406669.

[135] Y. Nam, S. Song, and J. Chung. “Clustered NFV Service Chaining Optimization in Mobile Edge Clouds”. In: *IEEE Communications Letters* 21.2 (2017), pp. 350–353. DOI: 10.1109/LCOMM.2016.2618788.

[136] Miroslaw Narbutt et al. “Adaptive VoIP Playout Scheduling: Assessing User Satisfaction”. In: *IEEE Internet Comput.* 9.4 (2005), pp. 28–34. DOI: 10.1109/MIC.2005.72. URL: https://doi.org/10.1109/MIC.2005.72.

[137] *Networking and Online Games – Understanding and Engineering Multiplayer Internet Games*. [Online; last accessed 01/2021]. 2006. URL: http://index-of.co.uk/Tutorials/Networking%5C%20And%5C%20Online%5C%20Games.pdf.

[138] NFV White Paper. “Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1”. Oct. 2012.

[139] Kathleen M. Nichols and Van Jacobson. “Controlling Queue Delay”. In: *ACM Queue* 10.5 (2012), p. 20. DOI: 10.1145/2208917.2209336. URL: https://doi.org/10.1145/2208917.2209336.

[140] Kathleen M. Nichols et al. “Controlled Delay Active Queue Management”. In: *RFC* 8289 (2018), pp. 1–25. DOI: 10.17487/RFC8289. URL: https://doi.org/10.17487/RFC8289.

[141] M C Nkosi and A A Lysko. “The use of P4 for 5G Networks”. In: *World Wireless Research Forum 40th meeting - WWRF40*. 2018, pp. 5–8.

[142] *P4 Consortium*. [Online; last accessed 01/2021]. 2021. URL: https://p4.org/.

[143] R. Pan et al. “PIE: A lightweight control scheme to address the bufferbloat problem”. In: *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*. 2013, pp. 148–155. DOI: 10.1109/HPSR.2013.6602305.

[144] Rong Pan et al. *Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem*. RFC 8033. Feb. 2017. DOI: 10.17487/RFC8033. URL: https://rfc-editor.org/rfc/rfc8033.txt.

[145] Aurojit Panda et al. “NetBricks: Taking the V out of NFV”. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 203–216. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/panda.

[146] F. Paolucci et al. “P4 Edge Node Enabling Stateful Traffic Engineering And Cyber Security”. In: *Journal of Optical Communications and Networking* 11.1 (2019), A84–A95. ISSN: 19430620. DOI: 10.1364/JOCN.11.000A84.

[147] Chrysa Papagianni and Koen De Schepper. “PI2 for P4: An Active Queue Management Scheme for Programmable Data Planes”. In: *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2019, Companion Volume, Orlando, FL, USA, December 9-12, 2019*. ACM, 2019, pp. 84–86. DOI: 10.1145/3360468.3368189. URL: https://doi.org/10.1145/3360468.3368189.

[148] István Pelle et al. “Towards Latency Sensitive Cloud Native Applications: A Performance Study on AWS”. In: *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*. Ed. by Elisa Bertino et al. IEEE, 2019, pp. 272–280. DOI: 10.1109/CLOUD.2019.00054. URL: https://doi.org/10.1109/CLOUD.2019.00054.

[149] C. Pham, N. H. Tran, and C. S. Hong. “Virtual Network Function Scheduling: A Matching Game Approach”. In: *IEEE Communications Letters* 22.1 (2018), pp. 69–72. DOI: 10.1109/LCOMM.2017.2747509.

[150] T. Pham and H. Chu. “Multi-Provider and Multi-Domain Resource Orchestration in Network Functions Virtualization”. In: *IEEE Access* 7 (2019), pp. 86920–86931. DOI: 10.1109/ACCESS.2019.2926136.

[151] Kjetil Raaen, Ragnhild Eg, and Carsten Griwodz. “Can gamers detect cloud delay?” In: *13th Annual Workshop on Network and Systems Support for Games, NetGames 2014, Nagoya, Japan, December 4-5, 2014*. Ed. by Yutaka Ishibashi and Adrian David Cheok. IEEE, 2014, pp. 1–3. DOI: 10.1109/NetGames.2014.7008962. URL: https://doi.org/10.1109/NetGames.2014.7008962.

[152] W. Rankothge et al. “Optimizing Resource Allocation for Virtualized Network Functions in a Cloud Center Using Genetic Algorithms”. In: *IEEE Transactions on Network and Service Management* 14.2 (2017), pp. 343–356. DOI: 10.1109/TNSM.2017.2686979.

[153] Ruben Ricart-Sanchez et al. “Hardware-Accelerated Firewall for 5G Mobile Networks”. In: *International Conference on Network Protocols - ICNP*. Vol. 2018-Septe. IEEE, 2018, pp. 446–447. ISBN: 9781538660430. DOI: 10.1109/ICNP.2018.00066.

[154] Ruben Ricart-Sanchez et al. “P4-NetFPGA-based network slicing solution for 5G MEC architectures”. In: *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2019*. IEEE, 2019, pp. 1–2. ISBN: 9781728143873. DOI: 10.1109/ANCS.2019.8901889.

[155] J. F. Riera et al. “TeNOR: Steps towards an orchestration platform for multi-PoP NFV deployment”. In: *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. 2016, pp. 243–250. DOI: 10.1109/NETSOFT.2016.7502419.

[156] J. F. Riera et al. “Virtual network function scheduling: Concept and challenges”. In: *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. 2014, pp. 1–5. DOI: 10.1109/SaCoNeT.2014.6867768.

[157] R. Riggio et al. “Scheduling Wireless Virtual Networks Functions”. In: *IEEE Transactions on Network and Service Management* 13.2 (2016), pp. 240–252. DOI: 10.1109/TNSM.2016.2549563.

[158] Fabricio Rodriguez, Christian Esteve Rothenberg, and Gergely Pongrácz. “In-network P4-based Low Latency Robot Arm Control”. In: *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2019, Companion Volume, Orlando, FL, USA, December 9-12, 2019*. ACM, 2019, pp. 59–61. DOI: 10.1145/3360468.3368178. URL: https://doi.org/10.1145/3360468.3368178.

[159] Christian Röpke and Thorsten Holz. “SDN Rootkits: Subverting Network Operating Systems of Software-Defined Networks”. In: *Research in Attacks, Intrusions, and Defenses*. Ed. by Herbert Bos, Fabian Monrose, and Gregory Blanc. Cham: Springer International Publishing, 2015, pp. 339–356. ISBN: 978-3-319-26362-5.

[160] R. Al-Saadi et al. “A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms”. In: *IEEE Communications Surveys Tutorials* 21.4 (2019), pp. 3609–3638. DOI: 10.1109/COMST.2019.2904994.

[161] Sahel Sahhaf et al. “Network service chaining with optimized network function embedding supporting service decompositions”. In: *Computer Networks* 93 (Oct. 2015). DOI: 10.1016/j.comnet.2015.09.035.

[162] Y. Sang et al. “Provably efficient algorithms for joint placement and allocation of virtual network functions”. In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057036.

[163] Stefan Savage et al. “TCP Congestion Control with a Misbehaving Receiver”. In: *SIGCOMM Comput. Commun. Rev.* 29.5 (Oct. 1999), pp. 71–78. ISSN: 0146-4833. DOI: 10.1145/505696.505704.

[164] Koen De Schepper and Bob Briscoe. *Explicit Congestion Notification (ECN) Protocol for Ultra-Low Queuing Delay (L4S)*. Internet-Draft draft-ietf-tsvwg-ecn-l4s-id-14. Work in Progress. Internet Engineering Task Force, Mar. 2021. 59 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-l4s-id-14.

[165] Koen De Schepper, Bob Briscoe, and Greg White. *DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)*. Internet-Draft draft-ietf-tsvwg-aqm-dualq-coupled-14. Work in Progress. Internet Engineering Task Force, Mar. 2021. 54 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-aqm-dualq-coupled-14.

[166] Koen De Schepper, Olivier Tilmans, and Bob Briscoe. *Prague Congestion Control*. Internet-Draft draft-briscoe-iccrg-prague-congestion-control-00. Work in Progress. Internet Engineering Task Force, Mar. 2021. 30 pp. URL: https://datatracker.ietf.org/doc/html/draft-briscoe-iccrg-prague-congestion-control-00.

[167] “SDNFV - OpenNetVM”. In: GitHub, 2020. URL: https://github.com/sdnfv/openNetVM.

[168] Roshan Sedar et al. "Supporting Emerging Applications With Low-Latency Failover in P4". In: *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies, NEAT@SIGCOMM 2018, Budapest, Hungary, August 20, 2018*. Ed. by Dipankar Raychaudhuri and Richard Li. ACM, 2018, pp. 52–57. DOI: 10.1145/3229574.3229580. URL: https://doi.org/10.1145/3229574.3229580.

[169] Vyas Sekar et al. "Design and implementation of a consolidated middlebox architecture". In: Apr. 2012, pp. 24–24.

[170] Stanislav Shalunov et al. *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817. Dec. 2012. DOI: 10.17487/RFC6817. URL: https://rfc-editor.org/rfc/rfc6817.txt.

[171] Sachin Sharma et al. "On Monolithic and Microservice Deployment of Network Functions". In: Apr. 2019. DOI: 10.1109/NETSOFT.2019.8806705.

[172] Amit Sheoran et al. "Contain-ed: An NFV Micro-Service System for Containing e2e Latency". In: Aug. 2017, pp. 12–17. ISBN: 978-1-4503-5058-7. DOI: 10.1145/3094405.3094408.

[173] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. "Misbehaving TCP Receivers Can Cause Internet-Wide Congestion Collapse". In: *Proceedings of the 12th ACM Conference on Computer and Communications Security*. CCS '05. Alexandria, VA, USA: Association for Computing Machinery, 2005, pp. 383–392. ISBN: 1595932267. DOI: 10.1145/1102120.1102170.

[174] Varun Singh and Jörg Ott. "Evaluating Congestion Control for Interactive Real-time Media". In: *WG, RMCAT Expires: August 04, 2014* (Jan. 2014).

[175] Nikolai Smolyanskiy and Mar Gonzalez-Franco. "Stereoscopic First Person View System for Drone Navigation". In: *Frontiers in Robotics and AI* 4 (Mar. 2017). DOI: 10.3389/frobt.2017.00011.

[176] T. Soenen et al. "Optimising microservice-based reliable NFV management orchestration architectures". In: *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*. 2017, pp. 1–7. DOI: 10.1109/RNDM.2017.8093034.

[177] Nathan Sousa et al. "Network Service Orchestration: A Survey". In: *Computer Communications* 142 (Mar. 2018). DOI: 10.1016/j.comcom.2019.04.008.

[178] Milan Stanojevic et al. "Can You Fool Me? Towards Automatically Checking Protocol Gullibility". In: *in HotNets '08, Seventh ACM Workship on Hot Topics in Networks*. Oct. 2008. URL: https://www.microsoft.com/en-us/research/publication/can-you-fool-me-towards-automatically-checking-protocol-gullibility/.

[179] Henrik Steen. "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management". In: *Masters Thesis, Dept of Informatics, Uni Oslo* (2017). URL: https://riteproject.files.wordpress.com/2018/07/thesis-henrste.pdf.

[180] Liyang Sun et al. "Optimal strategies for live video streaming in the low-latency regime". English (US). In: *27th IEEE International Conference on Network Protocols, ICNP 2019*. Proceedings - International Conference on Network Protocols, ICNP. 27th IEEE International Conference on Network Protocols, ICNP 2019 ; Conference date: 07-10-2019 Through 10-10-2019. IEEE Computer Society, Oct. 2019. DOI: 10.1109/ICNP.2019.8888127.

[181] L. Sundqvist. "Cellular Controlled Drone Experiment: Evaluation of Network Requirements". In: 2015.

[182] Mirko Suznjevic, Ivan Slivar, and Lea Skorin-Kapov. "Analysis and QoE evaluation of cloud gaming service adaptation under different network conditions: The case of NVIDIA GeForce NOW". In: *Eighth International Conference on Quality of Multimedia Experience, QoMEX 2016, Lisbon, Portugal, June 6-8, 2016*. IEEE, 2016, pp. 1–6. DOI: 10.1109/QoMEX.2016.7498968. URL: https://doi.org/10.1109/QoMEX.2016.7498968.

[183] *TCP extensions for long-delay paths*. RFC 1072. Oct. 1988. DOI: 10.17487/RFC1072. URL: https://rfc-editor.org/rfc/rfc1072.txt.

[184] Andrea Tomassilli. "Towards next generation networks with SDN and NFV". PhD thesis. June 2019.

[185] Roberto Viola et al. "QoE-based enhancements of Chunked CMAF over low latency video streams". In: *2019 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB 2019, Jeju, Korea (South), June 5-7, 2019*. Ed. by Huabo Sun. IEEE, 2019, pp. 1–6. DOI: 10.1109/BMSB47279.2019.8971894. URL: https://doi.org/10.1109/BMSB47279.2019.8971894.

[186] S. Vrontis, S. Xynogalas, and E. Sykas. "Steiner tree compilation of multicast under differentiated services constraints". In: *Journal of Communications and Networks* 9.1 (2007), pp. 84–92. DOI: 10.1109/JCN.2007.6182817.

[187] Peng Wang et al. "Dynamic function composition for network service chain: Model and optimization". In: *Computer Networks* 92 (2015). Software Defined Networks and Virtualization, pp. 408–418. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.07.020. URL: https://www.sciencedirect.com/science/article/pii/S1389128615003266.

[188] Y. Wang et al. "From Design to Practice: ETSI ENI Reference Architecture and Instantiation for Network Management and Orchestration Using Artificial Intelligence". In: *IEEE Communications Standards Magazine* 4.3 (2020), pp. 38–45. DOI: 10.1109/MCOMSTD.001.1900039.

[189] David Wetherall, Neil Spring, and David Wetherall. *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*. RFC 3540. June 2003. DOI: 10.17487/RFC3540. URL: https://rfc-editor.org/rfc/rfc3540.txt.

[190] Lisong Xu, Khaled Harfoush, and Injong Rhee. "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks". In: *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004*. IEEE, 2004, pp. 2514–2524. DOI: 10.1109/INFCOM.2004.1354672. URL: https://doi.org/10.1109/INFCOM.2004.1354672.

[191] Yang Xu et al. "Video Telephony for End-Consumers: Measurement Study of Google+, iChat, and Skype". In: *IEEE/ACM Trans. Netw.* 22.3 (2014), pp. 826–839. DOI: 10.1109/TNET.2013.2260354. URL: https://doi.org/10.1109/TNET.2013.2260354.

[192] Bo Yi et al. "A Comprehensive Survey of Network Function Virtualization". In: *Computer Networks* 133 (Mar. 2018). DOI: 10.1016/j.comnet.2018.01.021.

[193] Yu Yinbo et al. "Joint optimization of service request routing and instance placement in the microservice system". In: *Journal of Network and Computer Applications* 147 (Sept. 2019), p. 102441. DOI: 10.1016/j.jnca.2019.102441.

[194] S. Q. Zhang et al. "Joint NFV placement and routing for multicast service on SDN". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 2016, pp. 333–341. DOI: 10.1109/NOMS.2016.7502829.

[195] Wei Zhang et al. "Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization". In: *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '16. Irvine, California, USA: Association for Computing Machinery, 2016, pp. 3–17. ISBN: 9781450342926. DOI: 10.1145/2999572.2999602. URL: https://doi.org/10.1145/2999572.2999602.

[196] Wei Zhang et al. "OpenNetVM: A Platform for High Performance Network Service Chains". In: *Proceedings of the 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. HotMIddlebox '16. Florianopolis, Brazil: Association for Computing Machinery, 2016, pp. 26–31. ISBN: 9781450344241. DOI: 10.1145/2940147.2940155. URL: https://doi.org/10.1145/2940147.2940155.

[197] W. Zhijun et al. "Low-Rate DoS Attacks, Detection, Defense, and Challenges: A Survey". In: *IEEE Access* 8 (2020), pp. 43920–43943. DOI: 10.1109/ACCESS.2020.2976609.

[198] Ao Zhou et al. "LMM: latency-aware micro-service mashup in mobile edge computing environment". In: *Neural Computing and Applications* 32 (Oct. 2020). DOI: 10.1007/s00521-019-04693-w.

[199] Zhibin Zuo et al. "P4Label: packet forwarding control mechanism based on P4 for software-defined networking". In: *Journal of Ambient Intelligence and Humanized Computing* 0123456789 (2020). ISSN: 18685145. DOI: 10.1007/s12652-020-01719-3. URL: https://doi.org/10.1007/s12652-020-01719-3.