Received: 1 September 2018 | Revised: 22 March 2019 | Accepted: 3 May 2019

DOI: 10.1002/nem.2079

RESEARCH ARTICLE

WILEY

Efficient allocation of elastic interfederation encrypted streaming sessions

Rafael Xavier[1] | **Lisandro Zambenedetti Granville**[2] | **Filip De Turck**[1] | **Bruno Volckaert**[1]

[1]IDLab, Ghent University—imec, Ghent, Belgium

[2]Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

Correspondence
Rafael Xavier, IDLab, Ghent University—imec, Ghent, Belgium.
Email: rafael.xavier@ugent.be

Present Address
Technologiepark-Zwijnaarde 15, B-9052 Ghent, Belgium

Summary

Audio and video (A/V) collaboration platforms often use Internet cloud technologies to ensure elasticity. They generally operate on a best-effort basis, without quality or delivery guarantees. However, such guarantees are a premise of business-focused platforms, which often rely on static/dedicated infrastructure and hardware-based components. This article presents results obtained in the final stage of the Elastic Media Distribution (EMD) project, which targets the migration of a business-focused hardware-based A/V collaboration tool towards a more elastic, reliable, and secure cloud-based model. The use case under investigation is an educational scenario: teachers and students located at distributed sites collaborate under different data encryption policies. An existing model of collaboration streaming is extended to accommodate encryption-enabled streaming components, and new resource allocation heuristics are proposed to deploy these components under stringent service level agreement (SLA) constraints. An extended version of our evaluation framework, based on the CloudSim simulator, manages encryption-enabled components. A resource usage dataset was obtained by prototyping selected streaming components and evaluating their performance on the Virtual Wall large-scale test bed. This dataset is fed into the extended simulation framework. Simulation results show longer than expected delays when loading streaming components, an issue that jeopardises the user experience that can be alleviated by the algorithms proposed in this article. Results show that the proposed algorithms enable policy-based secured communications under bandwidth and virtual machine (VM) cost increases of 48% and 23%, respectively, if compared with a nonencrypted previous solution, and with set-up times remaining under the required 2-second deadline.

1 | INTRODUCTION

Digital collaboration solutions, in the past only available in-company, are now more widely adopted. Platforms like WhatsApp,[1] Facebook Messenger,[2] and Google Hangouts[3] efficiently use the elastic cloud infrastructure, automatically scaling to massive amounts of users. However, they are reached through best-effort Internet, without guarantees and prone to delays and congestion. These issues are mitigated in enterprise-wide solutions such as Skype for Business,[4] Cisco Unified Communications,[5] and Barco collaborative learning solutions.[6] However, these solutions control congestion problems by delimiting the architecture to private/on-premises infrastructure, where they often rely on nonelastic

 © 2019 John Wiley & Sons, Ltd.

in-house installed appliances and quality of service (QoS) rules enforced by dedicated network infrastructure, usually dependent on manual maintenance efforts.[7,8]

The EMD Project[9] designed a solution that combines reliability and elasticity, in a secure way, in one distributed audio and video (A/V) professional collaboration platform that focused on remote learning. Policy-based security is one of the goals, and a risk-based partitioning of the infrastructure in distinct security zones enables the enforcement of specific protection policies. The scenario is exemplified in Figure 1. Tutor A/V data are pictured by red orbs and are processed and distributed among all platform users. Endpoints have different capabilities of sourcing and receiving A/V data, and one source can provide data to multiple sinks. For instance, one endpoint expects 720p A/V input, another endpoint expects 480p A/V input, while the source encodes it in 1080p, potentially using different codecs (A/V compression standards) as well . Software components able to transcode/split data must be efficiently provisioned between source and destination endpoints, transforming the data in terms of A/V codec and resolution, matching the endpoints' capabilities.

Interaction streams sent back by students appear as green orbs and arrows, which together make up a single video stream that can show students on a single screen (represented by the yellow arrows and orbs and distributed to a selected group of users). Two security risk zones are defined: A safe zone has the grey dotted line set as boundary, and the rest is considered unsafe (with a higher risk of intercepted or compromised data). Therefore, an encryption policy is enforced to protect all data in transit out of the safe zone. The dotted arrows represent protected (encrypted) data flows. This article focuses on the educational scenario, but other scenarios have similar workflows and collaboration needs. Some of these are remote monitoring and crisis management (control rooms, tackled by the EMD research project), business meetings, and remote surgeries. Therefore, w.r.t. protecting the confidentiality of A/V data by enforcing encryption where necessary, the outcomes reported in this article can be extended to scenarios beyond the educational landscape.

Previous work analysed a similar educational scenario, proposing streaming model and resource allocation algorithms, and presenting service level agreement (SLA)/budget-oriented results.[10] In the proposed approach, teachers present classes to local and dynamically distributed remote students, who join and leave the collaboration, and A/V streams are exchanged among them through dynamically instantiated and configured platform components. The CloudSim framework[11] is used for evaluation. During the simulation, network, virtual machines (VMs) usage, and budget are monitored, aiming to reduce costs while keeping SLAs control. The allocation of encrypted streams, a potential need for intrafederation or Internet communication security policies, was not supported. In this article, we propose an extended approach allocating encrypted streams over a cloud infrastructure, with new models, algorithms, and a more realistic dataset based on measurements of a prototyped instance of the framework running on our Virtual Wall test bed.[12] Results showcase the final outcome of 2 years of research in the EMD project.

In order to provide secured services, extra stream and component models are proposed (encryption-enabled streaming processing software components). Then, resource allocation algorithms provision the required stream workflow by three different policies: fully encrypted, partially encrypted (protecting only predefined unsafe zones), and nonencrypted. The proposed algorithms are simulated, and results are compared with the previously detailed SHARED algorithm.[10]

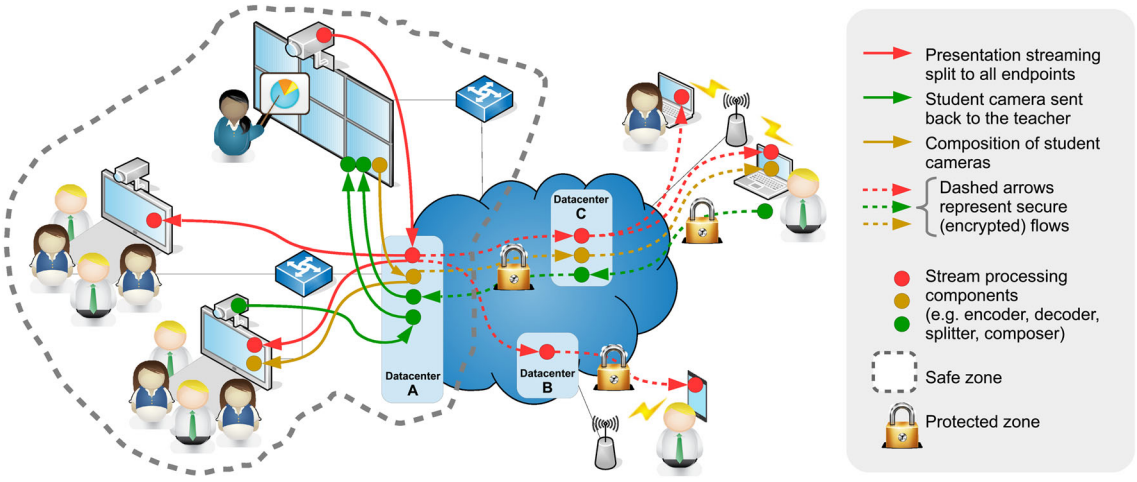

FIGURE 1 Example of the distributed educational scenario studied in this article, including topology (enterprise networks connected to the cloud), endpoints, cloud infrastructure, platform software components (orbs), and secure A/V transmissions between users (dashed when encrypted). The composed stream (yellow orbs and arrows) is the interaction streams sent by students to the teacher

Since no resource usage and imposed processing delay dataset was available for encrypted media processing components, we evaluated prototyped encrypted streaming components (using the GStreamer[13] library). Then, multiple A/V streams were placed over a set of servers hosted on the Virtual Wall[12] test bed infrastructure. The measurements have been stored in a complete per component resource usage dataset, eg, central processing unit (CPU) usage, memory usage, and imposed processing delay. This dataset was required to seed the simulator, used for more large-scale evaluations of component placement and configuration algorithms. Additionally, this new dataset has shown another issue: the algorithms proposed before[10] tolerated stream interruptions when inserting new components in an existing stream workflow (eg, a splitter in between an already streaming encoder-decoder pair). Despite being irrelevant when dealing with minimal delays, the new data have shown these interruptions as harmful for the user experience, a problem handled by the algorithms proposed in this article.

This article is structured as follows. In Section 2, we describe the related work in this area. Project requirements and models are detailed in Section 3. Then, in Section 4, we present the proposed resource allocation algorithms. In Sections 5 and 6, we present the prototyping and simulation results, respectively, and then, in Section 7, we finalise by describing our current conclusions and potential avenues for future research.

2 | RELATED WORK

In the studied educational scenario, several teachers and students are dynamically and geographically placed in different locations. Collaboration data are expected to flow in a one-to-many fashion and among endpoints with distinct processing capacities, as stated in Section 1. Therefore, software components that are able to compose and adapt these collaboration flows must be flexibly allocated along the cloud infrastructure. They must interconnect endpoints under acceptable time frame and costs and under appropriate security policies.

2.1 | Location-aware security policies

Distributed cloud computing and mobility solutions brought freedom and flexibility to the user. However, security concerns have also risen (eg, risks of transmitting over the Internet and processing/storing data externally). There are several cases of online data exposure, including data hijacking or eavesdropping when using vulnerable Wi-Fi access points, Internet home connections, and external routing backbones.[14-18] Therefore, risk levels can be associated with where the data are stored and processed, and several studies take location into account when proposing security solutions. Encryption and access-control services were proposed to tackle location-based security decisions.[19-21] However, they do not tackle the placement of security-enabled streaming components w.r.t. data location and associated risks.

In this article, risks are handled as recommended in the ISO 27005 standard,[22] which defines it as the combination of the likelihood and the impact of a security incident. In our scenario, we relate the likelihood to the data location, as selected infrastructure spots may be more vulnerable and harder to protect. As the enforcement of an end-to-end encryption mechanism in the application layer creates an additional barrier for attackers,[23] we propose location-based security policies. These enforce streaming components to encrypt the data when necessary, minimising associated business risks only when they are at unacceptable levels.

Many other valid efforts aim to protect user data by enforcing encryption on web services and streaming.[24,25] Other works aim to detect new vulnerabilities and improve protection mechanisms.[26] This article does not focus on advancing the state-of-the-art in secure data transmission but on the intelligent and autonomous placement of components that use state-of-the-art encryption mechanisms in the cloud. We compose end-to-end encrypted data flows under the guidance of security policies and enforced by the proposed component provisioning algorithms.

2.2 | Cloud resource allocation

Cloud infrastructures provide flexibility, and various efforts have focused on cloud resource allocation. Scenarios ranging from a single cloud data centre to multicloud deployments were studied.[27] They exploited elasticity and scalability but did not take into account the location of endpoints.[28-31] In the scenario studied in this article, endpoints are placed outside the cloud data centre (eg, mobile/home clients), using public/private networks. Therefore, one must consider how they reach the cloud data centres, as this can have a large impact on the user experience.

Another important requirement is the placement of new A/V streaming flows in acceptable time: project requirements imposed by the industrial partners in the EMD research project mandate a 2-second maximum user request response

(requirement explained in Section 3). This response time highly depends on the network performance and on the reservation algorithm's efficiency (resources like VMs and software components must be provisioned in acceptable time). Network function virtualisation (NFV)[32] offers complex network services by allocating, configuring, and chaining multiple network functions.[33,34] Nonetheless, NFV-based management systems focus on generic network functions being allocated in a single federation, while our approach and proposed algorithms focus on A/V specialised components composing workflows across multiple federations (eg, a stream originated in a location, the teacher room, split and transcoded in a second location, a cloud data centre, and delivered to an endpoint placed in a third location, a remote classroom).

Therefore, we needed a solution that focuses on placing specialised A/V processing components according to security policies, is able to interconnect endpoints placed out of the cloud, and is capable of establishing A/V collaboration under 2 seconds.

2.3 | Cloud-based A/V services allocation

Researchers specifically focused on placing multimedia conferencing services in the cloud. Named adaptive and dynamic scaling (ADS) mechanism, a scalable and elastic solution was proposed according to the number of conference participants, targeting smaller scenarios, such as distance learning, to massively multiplayer online game (MMOG) platforms managing millions of users.[35] Aiming to provide an acceptable user experience (eg, lower waiting times) at reduced costs, optimised media service selection and scheduling algorithms were proposed.[36] Nevertheless, these efforts do not tackle the specialised components, neither the composition of workflows across multiple federations or the 2-second deadline imposed by the EMD project.

Approaches to deal with these professional A/V collaboration needs of the EMD project have been proposed. A similar scenario has been studied, proposing encoding, transcoding, and decoding stream components.[37] The CloudSim[11] simulation framework was extended to support new features, eg, bandwidth monitoring, elastic multimedia workflow placement, and VM preemptive allocation, and used for evaluation purposes. A more elaborate preemptive VM management technique was also proposed,[38] using historical data and demand-based preemption to manage and populate a pool of running VMs. As a VM can take several minutes to be fully instantiated,[39] the preemptive instantiation of VMs is important to reduce the time to establish a new A/V streaming flow while fulfilling the 2-second deadline requisite. More focused on the placement of more complex component workflows, other research works[10,40,41] proposed new components, modelling more accurately the scenario. However, these efforts did not consider data protection. They also relied on estimated/extrapolated resource usage data sets as basis for larger-scale evaluation simulations, which were not available for the security-enabled components. This need motivated us to prototype and evaluate these components and obtain a dataset that is able to feed our simulation framework with more realistic usage information.

As mentioned in Section 1, another issue showed up when developing the work presented in this article: the existing approaches[10,40,41] tolerated stream reestablishments when composing one-to-many fashioned stream workflows. Figure 2 shows an example: on (1), the data from an encoder *A* is transcoded and delivered to a decoder *B*. While streaming from *A* to *B*, another decoder joins and requires the same stream provided by *A*. So, the stream between *A* and *B* is interrupted, as shown in (2), to insert a new splitting component, called tee (*T*). The final workflow is shown in (3), with *B* and *C* receiving the stream replicated by *T*. In previous work, we assumed a low delay when reestablishing an already flowing A/V stream and simulated components able to quickly restate them. However, the measurements obtained from the prototyped components, especially the security-enabled ones, showed nontolerable values, and we had to change the approach used when splitting one stream flow among multiple sink endpoints: The algorithms proposed in this article have been designed to take this into account while remaining interruption-free.

2.4 | Security-enabled A/V streaming components

The goal of including encryption in this study is to understand the impact of such functionality on the resource requirements for professional A/V collaboration solutions. Several options are available for streaming encryption. Considering Real Time Protocol (RTP) streaming sessions, some approaches can be adopted as protection: RTP over Transport Layer Security (TLS), RTP over Datagram Transport Layer Security (DTLS), Secure Real-time Transport Protocol over Datagram Transport Layer Security (DTLS-SRTP), Internet Protocol security (IPSEC) tunnels, among others.[42]

WebRTC[24,43] is another option, an architecture that enables secure A/V conferencing over the Internet. It contains authentication and signalling services, using Session Description Protocol (SDP) to negotiate session parameters, DTLS to negotiate encryption keys, and Secure Real-time Transport Protocol (SRTP) to protect the data between pairs of endpoints. Secure channels are established using Advanced Encryption Standard (AES), Secure Hash Algorithms 1 (SHA-1),

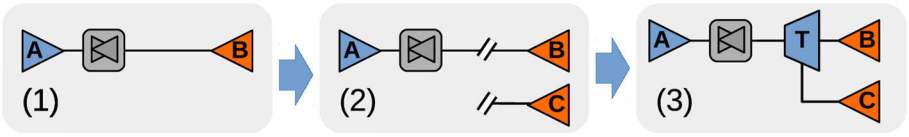

FIGURE 2 Example of a splitting component, named tee (T), being inserted between components from an existent streaming workflow. In (1), an encoder A streams to a decoder B, with a transcoder placed between them. Then, a decoder C is placed to also receive A's streamed data (simultaneously). To make T's inline insertion feasible, the link to B is interrupted, as shown in (2). Then, T is linked to A and B, reestablishing the stream flow (3)

and Secure Sockets Layer (SSL) certificates. WebRTC endpoints are implemented using the WebRTC Javascript API, a technology already embedded in modern web browsers. However, as WebRTC is initially designed for one-to-one communication, more efficient one-to-many features must be built to reach multiple users and avoid complex/costly mesh networks.[44]

This article is part of an effort to come up with the architecture of a new elastic A/V collaboration platform for multiple endpoints, with intelligent allocation of security-enabled components. Therefore, we focus on DTLS-SRTP to build secure A/V streaming components and provide a flexible model integrating these features. We measure the resource impact of the prototyped security-enabled components, to then seed large-scale simulations with this more realistic dataset. Then, we propose automated software component placement and configuration mechanisms that extend the previous solutions. These mechanisms optimise budget constraints and avoid stream interruptions while keeping imposed security requirements in check. The requirements and concepts used to define them are detailed in the next section.

3 | SCENARIO AND STREAM CONCEPTS

A/V streams, platform software components, and the requirements and concepts used to model the scenario are described in this section.

3.1 | EMD project requirements

The first EMD project requirement this article focuses on is the time taken to join a stream session: when a user wishes to join an A/V collaboration session (eg, by clicking in a join button in the endpoint interface), the platform is expected to handle more than 99% of the received requests in less than 2 seconds.

A second requirement is data protection. The establishment of secure remote A/V transmission of critical data and collaboration over public/private networks is required, and policy-based solutions a project choice to solve this issue. This article focuses on location-wise policies, providing different security levels according to where (in the network) the data are passing through. The proposed resource allocation algorithms mirror these policies. Finally, the costs of cloud and network resources are key to the economic viability of the proposed platform and must be kept in check. It is expected that compliance with specific project requirements (eg, projects with stringent delay requirements) will mandate trade-offs that can increase costs significantly.

3.2 | Stream data concepts

Video *codecs* are compression/decompression algorithms that implement video standards. In this article, we represent as $C \leftarrow (c_1 \dots c_n)$ the set of n available codecs (eg, h.264 AVC, h.265 HVEC, raw). Another concept, *data format*, is the combination of a codec, a video resolution, and an encryption function. So, a data format $d \leftarrow (\alpha, \beta, \gamma, \delta)$ represents the $\beta \times \gamma$ screen resolution encoded with codec α and encrypted by means of encryption algorithm δ. The encryption function represents the security algorithms applied after encoding the content for authentication and encryption, which include AES, SHA-1, no encryption (when no protection is needed), among many others possibilities. Additionally, in this article, *raw* is used to represent nonencoded data, and *plain* is used to represent the absence of encryption.

Codecs and data formats generically standardise the A/V information, so it can be transmitted and displayed. This information is defined as *content*, which combined with a *data format* composes a *stream data*. Therefore, K_d is the

stream data representation of a sourced content K using the data format d, *stream* is the transmission of a streaming data between two endpoints placed in the cloud, a source and a sink, and *stream workflow* is the set of all active streams being transmitted through the platform.

The stream workflow structure changes dynamically over time, when endpoints join and leave the streaming session, and the platform components described next are dynamically instantiated, configured, interconnected, and terminated to handle this morphic behaviour.

3.3 | Collaborative streaming workflow components

The processing of stream data is done by platform software components. A component is defined as two sets of stream data entries, one for input and another as output, and one internal stream data processing function, which defines the transformation of the data (ie, codec, change of resolution, generation of new content, replication or consolidation of flows, encryption of flows). For instance, a transcoding component C_T can have as input $I_T \leftarrow \{d_{(h.264,1920,1080,plain)}\}$, as output $O_T \leftarrow \{d_{(h.264,1440,720,DTLS-SRTP)}\}$, and an internal functionality $f_T(x) = y$ that for this example describes how this specific component converts a nonencrypted stream data into an encrypted and lower resolution version of it (from $d_{(h.264,1920,1080,plain)}$ to $d_{(h.264,1440,720,DTLS-SRTP)}$). Therefore, a component C_k is described as $C_k \leftarrow (I_k, O_k, f_k)$. The stream components used in this article are depicted in Figure 3 and described below.

Five main components are defined: *encoder*, *decoder*, *transcoder*, *tee* and *compositor*. Two auxiliary components are also depicted in Figure 3 and used when defining tee and compositor: *splitter* and *mixer*, respectively. Encoders convert raw streaming data formats into compressed/encrypted formats, while decoders convert compressed streaming data formats to raw. Transcoders implement the transformation between two data formats by changing codec (compression standard), resolution or encryption.

A splitter receives a nonencrypted flow and distributes it in a one-to-many fashion. This feature is used when composing a tee. Splitters can handle only one nonencrypted data format, and tees receive data in a stream data format and outputs it to many sinks potentially using distinct data formats for each of them. Therefore, the one-to-many distribution using different data formats performed by a tee is implemented by a set of transcoders chained with splitters (as shown in Figure 3). A similar concept is used when defining compositors.

Compositors receive multiple inputs of different contents and data formats and organise them in a single output stream, sometimes resizing or overlapping content. Each input is processed by a dedicated decoder, and all decoders' outputs are inserted into a mixer. A single output stream is provided by the mixer, which constitutes the compositor's output. The *composition control*interface determines how the single output stream is composed of the multiple inputs received (ie, how the video-in-video layout is arranged).

According to the EMD project, endpoints (ie, human end users manipulating these endpoints) can flexibly compose the A/V output displayed to the end user. Inputs can be omitted, overlapped, resized, or processed in many video-in-video ways. Additionally, compositors are allocated out of the cloud infrastructure, not affecting the proposed metrics for

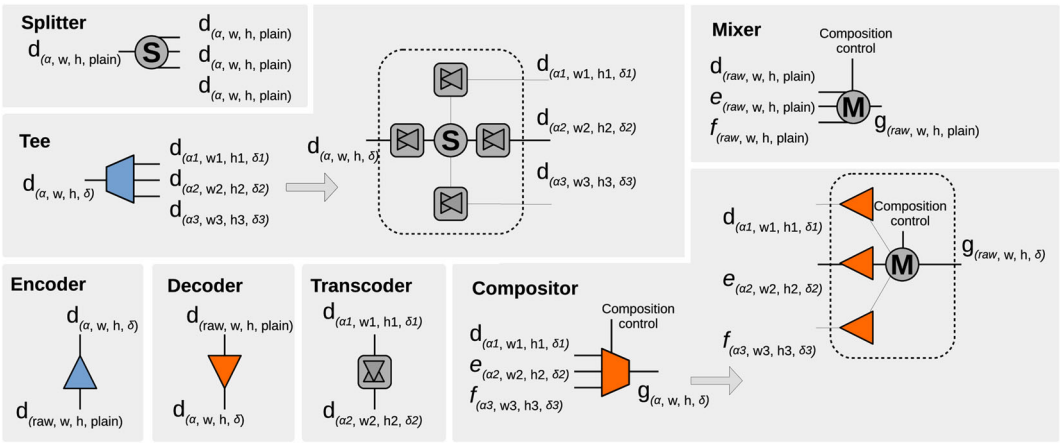

FIGURE 3 Components used to model A/V stream workflows. Tees, compositors, transcoders, encoders, and decoders are depicted with examples of input and output sets. In these examples, $d_{\alpha,\beta,\gamma,\delta}$ represents a content d encoded using the data format α, in a $\beta \times \gamma$ resolution, and under an encryption function δ. Absence of encryption is described as plain and nonencoded content as raw

cloud-related costs. Therefore, in this article, we focus on the resource usage impact of compositors up to the decoders that process their inputs and leave out the highly configurable mixer design.

3.4 | Stream workflow composition

Components are interconnected by the platform to compose chains (stream workflows). A connection between two components is possible when source output and sink input use the same data format. An example of a workflow of chained components is depicted on Figure 4. Encoders, decoders, tees, compositors, and transcoders are dynamically deployed (when each stream is requested) to transmit two contents (A and B) and display compositions of them in three distinct screens. This dynamic component allocation (and flexible workflow composition) is regulated by the resource allocation algorithms taking into account optimisation-driven design choices (depicted in Section 4).

A consequence of interconnections is the accumulated delay, which can be affected by components' complexity, adopted data formats, chosen resolution and compression standards (codecs), and duration of DTLS handshakes. We describe it in three subdivisions: (a) *component instantiation delay*, the time taken to make all requested components available for usage (requested, instantiated, and configured in parallel), (b) *processing delay*, the time the transmission of A/V frame takes from the source to the sink, and (c) *stream placement delay*, the sum of the two first ones, ie, the time between the user request and first frame received on the sink. Figure 5 pictures the placement delay for an example of stream workflow composed of four components: encoder, transcoder, tee, and decoder.

The stream placement delay ($D_{placement}$) is the sum of the processing delay ($D_{processing}$) and the component instantiation delay ($D_{instantiation}$). The processing delay is the sum of the processing delays of all components, plus the network delay among them. The instantiation delay is the maximum time for all required components to become available, as they are instantiated in parallel. This is the time the user waits after requesting a transmission: first, the components are instantiated and, when done, the transmission throughout all chained components takes place, reaching the user after all components process the data and the first A/V video frame is received.

This time is composed of the time the request takes to arrive to the platform, the VM instantiation time (reduced by employing an efficient VM allocation algorithm), and component instantiation/set-up time. So, for a chain composed of n interconnected components, from c_1 to c_n, with the network delay between two components a and b defined as $nd_{a,b}$, the component request time of component c defined as rt_c, the component VM instantiation time defined as vt_c, the component setup time defined as st_c, and the component processing delay defined as pd_c, $D_{placement}$ is given by Equation (1).

$$D_{placement} = D_{processing} + D_{instantiation} = pd_{c_1} + \sum_{i=2}^{n}(pd_{c_i} + nd_{c_{i-1},c_i}) + \max_{i\in(1..n)}(rt_{c_i} + vt_{c_i} + st_{c_i}). \quad (1)$$

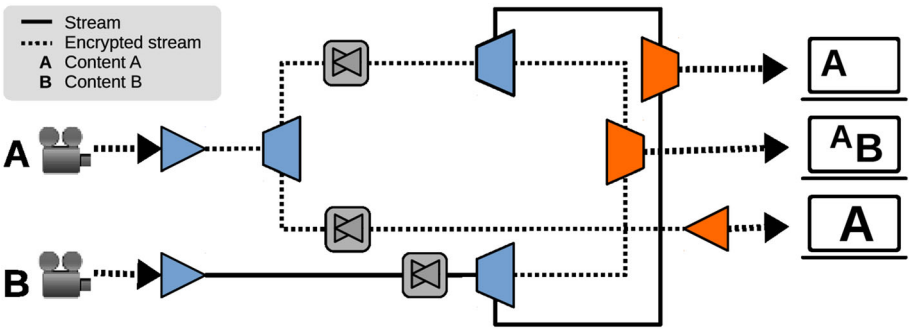

FIGURE 4 On this example, two distinct A/V sources (two camera/encoder pairs) are streamed, being split by tees and transmitted in different paths. Transcoders process the data on the way, while decoders and compositors prepare it to be displayed on the output screens

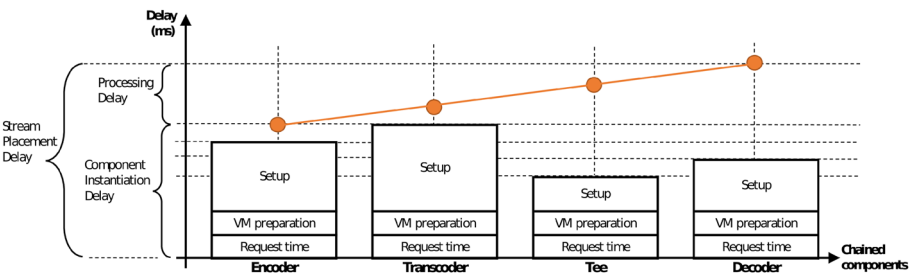

FIGURE 5 Example of the delay imposed when provisioning chained streaming components, composed of the delay to set them up and the processing delay

Before instantiating the components in the cloud infrastructure, the choice of where they will be placed and connected to the existing stream workflow must be taken. These are decisions taken by the component placement algorithms described in the next session.

4 | INTELLIGENT COMPONENTS ALLOCATION

Stream components must be placed and chained in a budget-wise fashion while enforcing security policies and SLA constraints, and streaming flows must not be interrupted and reestablished (issue described in Section 2). This section describes the algorithms proposed to deal with these demands.

4.1 | Location-aware security policies

The evaluated scenario is partitioned in a safe zone and an unsafe one, as explained in Section 1. Unsafe zones tend to be less protected and prone to higher risk levels than safe zones. The ISO 27005 standard[45] defines IT risk as the potential of having a threat harming the organisation by exploiting vulnerabilities of assets. Risk is measured combining the likelihood and the impact of an incident (eg, in terms of image or financial losses). For our scenario, assessed with focus on potential data interception leaks, Figure 6 shows initial risk and the residual risk (ie, risk level that remains after applying the security countermeasures) matrices, as explained next.

To compose the matrix model, likelihood and impact are split in five levels: from very unlikely (0) to frequent (4), and from very low (0) to very high (4), respectively. The sum of both provides risk scores from 0 to 8: up to 2, represented by the green cells, are considered low risk and negligible; higher than 5 (red cells) are ranked as high risks and not tolerable; and between 3 and 5 (yellow cells) as medium risks, with mandatory risk reduction or strict monitoring to trigger rapid remediation.

Our scenario has been risk assessed and mapped to such a risk matrix, based on the confidentiality of the data and the zones over which this data transits. With regards to the impact, the potentially compromised asset (the transmitted A/V information) can be tagged as *confidential* (eg, confidential data, new projects, data under nondisclosure agreements, and data that must be possessed only by authorised people), *internal* (eg, internal training courses, noncritical reports, and data that must stay inside the company boundaries), or *public*, for public reports, advertising projects, and other information that can be made public. The leak impact is mapped from these three tags to very high, medium, and very low, respectively. The likelihood of a security incident is based on the two already defined safe/unsafe zones: Safe zones have better managed security processes and tools, preventing or reacting faster to threats, and an incident is considered possible; however, unsafe zones have less security controls available, and attempts to compromise the system are expected to occur more often (eg, endpoints placed on the Internet are more prone to viruses and external networks are more susceptible to man-in-the-middle attacks). Aiming to reduce these risks, the planned countermeasure is the enforcement of encryption. Figure 6 shows our risk mapping and residual risks after this is enforced:

- **A and B:** public information transiting in safe or unsafe zones; low impact in the case of a data leak, as the information is already public. B is encrypted to avoid a medium risk, moving the likelihood of incidents to unlikely.

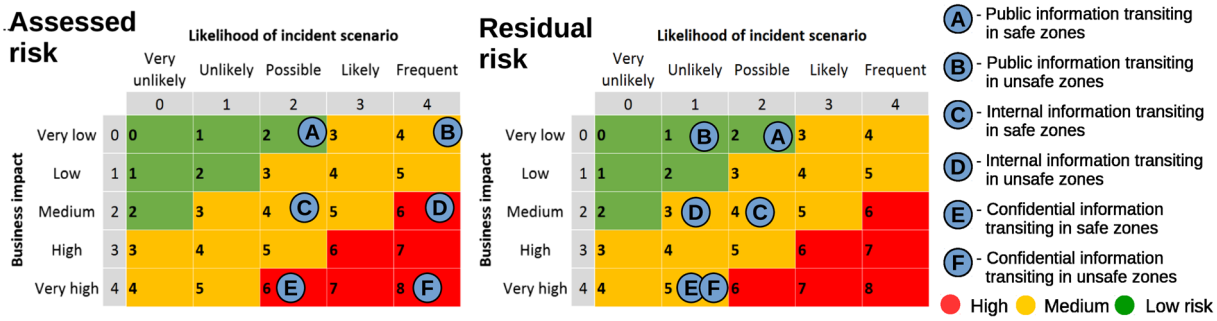

FIGURE 6 Matrices for the initial and residual risks w.r.t. collaborative information in transit through safe and unsafe zones. Six risks, from *A* to *F*, are mapped to three risk levels: high, medium, and low

- **C and D:** internal information must transit only internally, in the safe zone. There is no need for encrypting C. However, D represents a high risk, and its encryption moves the likelihood of incidents to unlikely, reducing the risk from the high to the medium level.
- **E and F:** confidential information must always be protected, irrelevant of being transmitted internally or externally. Encryption is enforced for both E and F, and the reduced likelihood of leaking the data migrates the risk from the high to the medium level.

Such a remediation plan, and the acceptance of the residual risk levels, must be approved in a business decision. Once decided, they can be translated to security policies and enforced within the organisation. Our modelled policies are (a) no encryption, (b) partial encryption, encrypting data only when using external links/Internet, and (c) total encryption, encrypting all data, even internally in the organisation, allowing access only to a selected group of authorised persons. Finally, these policies must be mapped into streaming component allocation algorithms to allow performing the policy-based establishment of encrypted or nonencrypted stream sessions.

Risks C, D, E, and F remain in the medium risk partition, requiring extra countermeasures such as monitoring and incident handling/remediation. Although crucial for the security framework, these extra controls are not in the scope of this article. Additionally, these information classification and risk policies are dependent on organisation peculiarities.

4.2 | Auxiliary algorithms and definitions

The definitions listed in Table 1 are used to describe the proposed component allocation algorithms: Non interrupter and not encrypted (NI), Non interrupter and partially encrypted (NI-PE), and Non interrupter and fully encrypted (NI-FE), which implement the proposed security policies *no encryption*, *partial encryption*, and *total encryption*, respectively. Other relevant definition used to list the algorithms is the auxiliary function $link(x, y, s, dc)$, shown in Algorithm 1. Its main functionality is to chain two components, x and y, in a one-to-one or one-to-many fashion, making y the next after x and x the previous before y. As components can be chained only when matching data format (a restriction detailed in Section 3.4), in case of data format incompatibility a transcoder is included to adapt data formats and make the interconnection feasible (lines 15 to 17).

4.3 | NI algorithm

This algorithm, listed in Algorithm 2, implements the nonencryption policy, described in Section 4.1. Initially, it searches for a compositor associated to the sink endpoint and attaches the new decoder to it. If it finds a compositor, a new com-

TABLE 1 Common functions for the proposed component allocation algorithms

Item	Definition
$x \Leftrightarrow y$	Connects two components x and y.
$dst(x)$	Returns the destination endpoint of a stream request x.
$d_{in}(x)$	Returns the input data format of a component x, or a set of data formats in case of more than one.
$d_{out}(x)$	Returns the output data format of a component x, or a set of data formats in case of more than one.
$ver_{crypto}(x)$	Returns the encrypted version of a data format x.
$ver_{plain}(x)$	Returns the nonencrypted version of a data format x.
$dc(x)$	Returns the data centre where a component x has been placed.
$bw(x)$	Returns the average bandwidth used when transmitting a stream data that uses a data format x.
$d_{src}(x)$	Returns the data format that will be sent by the source of a stream request x.
$d_{dst}(x)$	Returns the data format that will be expected by the destination of a stream request x.
$r_{src}(x)$	Returns the data centre closest to the source endpoint of a stream request x.
$r_{dst}(x)$	Returns the data centre closest to the destination endpoint of a stream request x.
$findComp(x)$	Search for a compositor associated with the destination endpoint x, returning $\varnothing$ if non existent.
$encoder(x, y)$	Creates an encoder instance at data centre y using data format x as output.
$decoder(x, y)$	Creates a decoder instance at data centre y using data format x as input.
$compositor(x)$	Creates a compositor at data centre x.
$tee(x, y)$	Creates a tee at data centre y using data format x for all inputs and outputs.
$transco(x, y, z)$	Creates a transcoder at data centre z using input data format x and output data format y.
$find(x, y, z)$	Searches for a component placed on data centre z, of type x and processing the content of a stream y.
$tieUp(x, y)$	Flags a component x as being used by a stream y.
$start(x)$	starts a stream request x.

positor is allocated (Algorithm 2, lines 3 and 4). Then, it searches along the stream workflow for a tee as the best spot to connect the new sink endpoint. Three distinct actions follow: (a) It does find a compatible tee near to the sink data centre, on line 8, and connects the endpoint to it; (b) it does not find a tee near the sink data centre, but finds one near the source data centre; then, it connects a new tee (placed in the sink data centre) to the found tee and connects the sink endpoint to the new tee (lines 11 and 15); (c) if no tee is found, the content is not being streamed yet; in this case, all required components are provisioned and started (lines 19 to 29). The overall aim is to reduce VM and bandwidth cost by reusing existent streams as near as possible to the sink endpoint and to reduce the streaming start delay by reusing already instantiated components, instead of instantiating new ones.

Algorithm 1 - LINK(x,y,s,dc) chains two components, x and y, in a one-to-one or one-to-many fashion. They are directly connected (line 6) when dealing with the same data format (line 5), otherwise interconnected by a transcoder that makes them compatible (line 17). To reduce bandwidth usage, the predicted usage levels are utilised to decide in which data centre to place the transcoder (line 9).

```
1:  function LINK(x, y, s, dc)
2:      x, y ← components to be linked
3:      s ← stream to be handled by x and y
4:      dc ← data centre to place a transcoder, or ∅ for auto placement
5:      if d_out(x) = d_in(y) then
6:          x ⟺ y
7:      else
8:          if dc! = ∅ then
9:              if bw(d_out(x)) < bw(d_in(y)) then
10:                 dc ← dc(y)
11:             else
12:                 dc ← dc(x)
13:             end if
14:         end if
15:         tra ← trans(d_out(x), d_in(y), dc)
16:         tieUp(tra, s)
17:         x ⟺ tra ⟺ y
18:     end if
19:     tieUp(y, s)
20:     tieUp(x, s)
21: end function
```

Algorithm 2 - Pseudocode for the NI algorithm - Firstly, a decoder is placed (line 6). Then, the algorithm looks for a tee to connect the decoder to (lines 8 and 11). First choice is a tee placed in the same data centre as the decoder (search in line 8), then one placed in the same data centre as the encoder (search in line 11). If no tee handling s is found, then the whole chain is allocated (lines 19-29). Initial component chains (one-to-one data flows) always include a tee, which is placed in line 25. This ensures that no interruptions will occur when subsequently deciding to deliver s to more than one decoder (one-to-many data flows).

```
1:  function NI(s)
2:      s ← stream request being allocated
3:      if !(comp ← findComp(dst(s))) then
4:          comp ← compositor(s)
5:      end if
6:      dec ← decoder(d_dst(s), r_dst(s))
7:      link(dec, comp, s, ∅)
8:      if tee_dst ← find("Tee", s, r_dst(s)) then
9:          link(tee_dst, dec, s, ∅)
10:     else
11:         if tee_src ← find("Tee", s, r_src(s)) then
12:             if r_src(s) = r_dst(s) then
13:                 link(tee_src, dec, s, ∅)
14:             else
15:                 tee_dst ← tee(s, d_dst(s), r_dst(s))
16:                 link(tee_src, tee_dst, dec, s, ∅)
17:             end if
18:         else
19:             enc ← encoder(s, d_src(s), r_src(s))
20:             tee_src ← tee(s, d_src(s), r_src(s))
21:             if r_src(s) = r_dst(s) then
22:                 link(enc, tee_src, s, ∅)
23:                 link(tee_src, dec, s, ∅)
24:             else
25:                 tee_dst ← tee(s, d_dst(s), r_dst(s))
26:                 link(enc, tee_src, s, ∅)
27:                 link(tee_src, tee_dst, s, ∅)
28:                 link(tee_dst, dec, s, ∅)
29:             end if
30:         end if
31:     end if
32:     start(s)
33: end function
```

Additionally, existing collaboration streams need to be free from interruption. Figure 7 exemplifies a stream (the first for this content, initialising a stream workflow) being placed according to the proposed algorithms and according to SHARED,[10] a previously proposed allocation approach subject to interruptions. SHARED places encoder, transcoder, and decoder in line, allocating less components than the others, but staying susceptible to an interruption if a new endpoint asks for the same content and a tee must be included within the running workflow. On the other hand, NI places tees at the start, even when handling just one endpoint, facilitating the addition of new sink endpoints. The same strategy is employed by NI-PE and NI-FE to ensure no streaming interruptions occur when potentially later on the number of endpoints changes.

4.4 | NI-PE algorithm

The NI-PE algorithm, which implements the partial encryption policy described in Section 4.1, is listed in Algorithm 3. Also interruption free, the main difference to NI is the encryption enforced when endpoints are in unsafe zones or the transmitted data flows over unsafe zones (eg, among data centres and over the Internet).

Algorithm 3 - Pseudocode for the NI-PE algorithm - As in the NI algorithm, a decoder is placed (line 6), then the algorithm looks for a tee to connect the decoder to (lines 8 and 11), and if no tee handling s exists, then the whole chain is allocated (lines 23-35). The main difference is the location-aware encryption policy enforcement when inter-data centre communication is detected. Lines 11 and 12 detect the need for connecting with a remote tee and introduce an encryption-enabled transcoder in line 17. The same logic is found in line 25 and 31: encoders and decoders placed in distinct data centres are detected and encryption-enabled transcoders are allocated if necessary. On the other hand, lines 13 and 26-27 connect local resources without enforcing encryption.

1: **function** NI-PE(s)
2: $s \leftarrow$ stream being allocated
3: **if** !($comp \leftarrow findComp(s)$) **then**
4: $comp \leftarrow compositor(r_{dst}(s))$
5: **end if**
6: $dec \leftarrow decoder(s, d_{dst}(s), r_{dst}(s))$
7: $link(dec, comp, s, \emptyset)$
8: **if** $tee_{dst} \leftarrow find(\text{“Tee"}, s, r_{dst}(s))$ **then**
9: $link(tee_{dst}, dec, s, \emptyset)$
10: **else**
11: **if** $tee_{src} \leftarrow find(\text{“Tee"}, s, r_{src}(s))$ **then**
12: **if** $r_{src}(s) == r_{dst}(s)$ **then**
13: $link(tee_{src}, dec, s, \emptyset)$
14: **else**
15: $tee_{dst} \leftarrow tee(s, d_{dst}(s), r_{dst}(s))$
16: $data \leftarrow ver_{crypto}(d_{in}(tee_{dst}))$
17: $tra \leftarrow transco(s, d_{out}(tee_{src}, data, r_{src}(s))$
18: $link(tee_{src}, tra, s, r_{src}(s))$
19: $link(tra, tee_{dst}, s, r_{dst}(s))$
20: $link(tee_{dst}, dec, s, \emptyset)$
21: **end if**
22: **else**
23: $enc \leftarrow encoder(s, d_{src}(s), r_{src}(s))$
24: $tee_{src} \leftarrow tee(s, d_{src}(s), r_{src}(s))$
25: **if** $r_{src}(s) = r_{dst}(s)$ **then**
26: $link(enc, tee_{src}, s, \emptyset)$
27: $link(tee_{src}, dec, s, \emptyset)$
28: **else**
29: $tee_{dst} \leftarrow tee(s, d_{dst}(s), r_{dst}(s))$
30: $data \leftarrow ver_{crypto}(d_{in}(tee_{dst}))$
31: $tra \leftarrow transco(s, d_{out}(tee_{src}, data, r_{src}(s))$
32: $link(tee_{src}, tra, s, r_{src}(s))$
33: $link(tra, tee_{dst}, s, r_{dst}(s))$
34: $link(tee_{dst}, dec, s, \emptyset)$
35: **end if**
36: **end if**
37: **end if**
38: $start(s)$
39: **end function**

Encryption-enabled transcoders, encoders, and decoders are used when necessary. As tees do not handle encrypted streams, auxiliary transcoders are attached to their input/outputs: They decrypt/encrypt the flows so tees can handle

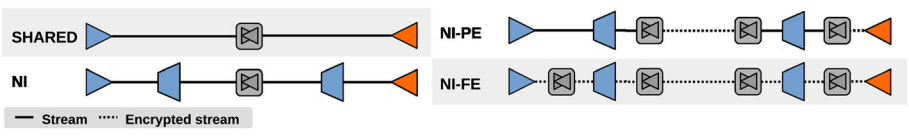

FIGURE 7 Example of components allocated in the provisioning of a first stream placed using the algorithms SHARED, NI, NI-PE, and NI-FE

them. For instance, a *tee-to-tee* connection through an unsafe zone is modelled as *tee-transcoder-transcoder-tee*, but a *tee-to-decoder* chain is modelled as *tee-transcoder-decoder* (as the decoder can handle encryption). The same is valid for *encoder-to-tee* pipelines, as encoders can handle encrypted streams too. To ensure that encryption/decryption operations occur only inside safe zones, these auxiliary transcoders are placed as close as possible to the tees, in the same data centre (these characteristics are depicted in Figure 7). The reason is that it is not possible for a tee to distribute DTLS-SRTP to multiple sinks (each sink must do a handshake), and to customise the encryption and video format of each output individually adds a lot of extra complexity in the tee component, which would need to be handled by the component and VM allocation algorithms (as each tee would essentially be a per-use case custom component).

4.5 | NI-FE algorithm

The last of the proposed algorithms, NI-FE is listed in Algorithm 4. Remaining interruption-free as NI and NI-FE, it implements the full encryption policy, meaning that all data streams must be encrypted. Therefore, as shown in Figure 7, encoders and decoders are always encrypted. As tees do not handle encrypted streams, their inputs or outputs always have a transcoder to encrypt/decrypt data and ensure the policy compliance.

Algorithm 4 - Pseudocode for the NI-FE algorithm - As in the NI and NI-PE algorithms, a decoder is placed (line 6), then the algorithm looks for a tee to connect the decoder to (lines 8 and 11), and if no tee handling s exists, than the whole chain is allocated (lines 23-35). The main difference is the encryption policy enforcement for the whole A/V streaming workflow. Lines 6 and 23 enforce it for decoders and encoders, while lines 16-17 and 30-31 guarantee that all tee outputs are encrypted by attached transcoders.

```
1:  function NI-FE(s)
2:      s ← stream being allocated
3:      if !(comp ← findComp(s)) then
4:          comp ← compositor(r_dst(s))
5:      end if
6:      dec ← decoder(s, ver_crypto(d_dst(s)), r_dst(s))
7:      link(dec, comp, s, ∅)
8:      if tee_dst ← find("Tee", s, r_dst(s)) then
9:          link(tee_dst, dec, s, ∅)
10:     else
11:         if tee_src ← find("Tee", s, r_src(s)) then
12:             if r_src(s) == r_dst(s) then
13:                 link(tee_src, dec, s, ∅)
14:             else
15:                 tee_dst ← tee(s, d_dst(s), r_dst(s))
16:                 data ← ver_crypto(d_in(tee_dst))
17:                 tra ← transco(s, d_out(tee_src, data, r_src(s))
18:                 link(tee_src, tra, s, r_src(s))
19:                 link(tra, tee_dst, s, r_dst(s))
20:                 link(tee_dst, dec, s, ∅)
21:             end if
22:         else
23:             enc ← encoder(s, ver_crypto(d_src(s)), r_src(s))
24:             tee_src ← tee(s, d_src(s), r_src(s))
25:             if r_src(s) = r_dst(s) then
26:                 link(enc, tee_src, s, ∅)
27:                 link(tee_src, dec, s, ∅)
28:             else
29:                 tee_dst ← tee(s, d_dst(s), r_dst(s))
30:                 data ← ver_crypto(d_in(tee_dst))
31:                 tra ← transco(s, d_out(tee_src, data, r_src(s))
32:                 link(tee_src, tra, s, r_src(s))
33:                 link(tra, tee_dst, s, r_dst(s))
34:                 link(tee_dst, dec, s, ∅)
35:             end if
36:         end if
37:     end if
38:     start(s)
39: end function
```

5 | COMPONENTS PROTOTYPING

This article propose resource allocation algorithms for encryption-enabled streaming components. To simulate and evaluate these algorithms on a larger scale, a dataset with realistic resource usage is needed. In this section, the prototyping and evaluation of components are explained.

5.1 | Virtual Wall test bed

The Virtual Wall[12] is an extensive experiment environment with hundreds of nodes interconnected via gigabit ethernet switches. Multiple network topologies can be created among the nodes, customising, eg, delay, packet loss, and bandwidth.

VMs and dedicated servers can be dynamically allocated, automatically installed, and remotely accessed immediately, facilitating research experiments. In a larger scope, it is integrated within the North American GENI[46] and the European Fed4Fire+[47] initiatives. This test bed was used to evaluate the prototyped components.

5.2 | Video properties

For evaluation purposes, videos are generated and received in four distinct resolution scales: 256×144, 640×360, 1280×720, and 1920×1080, which is a set that can be extended. Streams are encoded using the H.264 AVC codec, or kept unmodified (raw), and transported by RTP. Additionally, DTLS-SRTP (AES with 128-bit keys and 80-bit HMAC-SHA-1 authentication) encryption is used instead of RTP, where applicable.

5.3 | Prototyped components

Encoders, decoders, transcoders, and tees were prototyped using the C programming language and the GStreamer[13] library. The source code, and the evaluated components' runtime evaluation statistics are availabe online.[48*] The encoder reads raw A/V files and can be parameterized in terms of resolution and frames per second. It encodes the data using H.264 AVC and sends it through RTP/DTLS-SRTP to the next component in the stream workflow. The decoder performs the opposite operation: receive the encoded/encrypted data, unpacks the RTP/DTLS-SRTP, decrypts it if encrypted, and provides displayable raw A/V data to a sink (eg, a screen and a compositor). Transcoders are configurable to receive and provide any combination of any non-raw video format, using RTP and DTLS-SRTP, and have a data scaler at the component's core (eg, it can receive 1920×1080 and provide a 640×480 transformed output). Complementing the prototyped components, tees only handle RTP streams and are unable to process raw or encrypted data. This is a design decision, motivated by two issues: (a) tees are always placed in the middle of a streaming workflow and therefore always receiving non-raw encoded data; (b) Gstreamer's DTLS-SRTP negotiations are one-to-one fashioned, and as such, splitters are unable to replicate encrypted flows to multiple sinks. However, one transcoder can be attached to each output to perform this negotiation individually, and the placement of these transcoders within the tee component or outside them, as new components, is a design choice. We decided to use external transcoders because this reduces the amount of tee variations to be evaluated and preemptively kept available in a component pool by the VM allocation algorithms.

Shown in the components set (Figure 3), compositors were not prototyped, as they are part of the endpoint solution and therefore not placed in Cloud nodes. Endpoints are the platform clients, a customised software application that runs in the end user workstation or mobile client, and this article focuses exclusively on the resource usage in the cloud. Decoders and encoders are also allocated outside of the cloud, but were prototyped as they are mandatory to complete the end-to-end workflow and make the evaluation feasible.

5.4 | Evaluation methodology

The prototyped components were orchestrated and evaluated using the Virtual Wall infrastructure, organised as shown in Figure 8. VMs are interconnected by point-to-point IP connections, with a realistic delay set-up, and an additional VM meant to send orchestration commands and collect statistics is connected to all other VM using a dedicated and low delay management network. Each VM allocated to these experiments consisted of an Intel Xeon CPU with 2.4 GHz (6 cores), 16 GB of RAM, and at least 50 GB of storage, running Ubuntu 16.06. Depending on the component being evaluated, the workflows were composed as shown in Figure 8, whose items from (1) to (3) show the order the components were placed over the evaluation test bed resources:

1. Encoders and decoders were evaluated using two VMs, one for the encoder and one for the decoder. Encoder-decoder pairs were instantiated for all available video formats listed on Section 5.2.
2. Splitter evaluation used three VMs, one for the encoder, one for the splitter, and one to host multiple dummy decoders. Encoders and decoders were measured in (1) and acted as auxiliary components, as the splitter was the main focus of this evaluation step.
3. Transcoders were evaluated using three VMs, each one hosting a single component: an encoder, a transcoder, and a decoder. As a transcoder transforms the video data between two data formats, workflows for all combinations of data formats listed in Section 5.2 were evaluated.

*Source code will be made publicly available upon acceptance of this work.

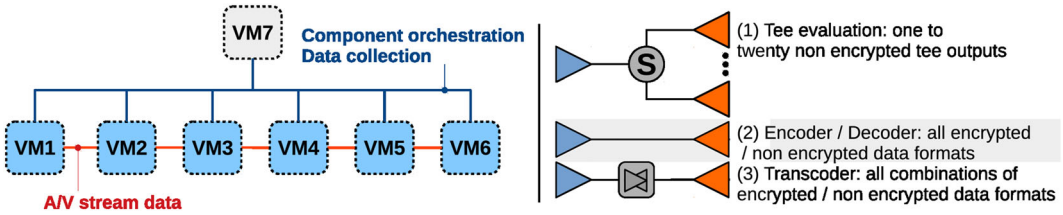

FIGURE 8 On the left side, the Virtual Wall V Ms used to evaluate the components, connected by the red lines, on which the streaming data is transmitted (from VM1 to VM6). Additionally, an extra VM7, connected to a low delay management network (depicted by the blue lines), handles statistic collection and orchestration through automatic remote commands. On the right side of this illustration, the three component combinations used for evaluation are shown: (1) one splitter, one encoder, and multiple decoders, composing a workflow used to evaluate tees with multiple outputs; (2) encoder/decoder pairs for encoder/decoder evaluation; and (3) encoder/transcoder/decoder compositions used to evaluate transcoders

TABLE 2 Bandwidth average usage per A/V setup for encrypted and nonencrypted streams

A/V Set-up	Encryption	Average	Standard Deviation
256 × 144 H264 AVC	None	0.6713 Mbit/s	0.0
256 × 144 H264 AVC	DTLS-SRTP AES[128 bits] SHA-1[80 bits]	1.2798 Mbit/s	0.0208
640 × 360 H264 AVC	None	2.8268 Mbit/s	0.0
640 × 360 H264 AVC	DTLS-SRTP AES[128 bits] SHA-1[80 bits]	3.3621 Mbit/s	0.0581
1280 × 720 H264 AVC	None	9.9559 Mbit/s	0.0
1280 × 720 H264 AVC	DTLS-SRTP AES[128 bits] SHA-1[80 bits]	10.4003 Mbit/s	0.1122
1920 × 1080 H264 AVC	None	18.2087 Mbit/s	0.0639
1920 × 1080 H264 AVC	DTLS-SRTP AES[128 bits] SHA-1[80 bits]	18.3463 Mbit/s	0.2275

Tees and compositors were not directly prototyped and evaluated: as shown in Figure 3, tees are composed of transcoders and splitters, which were measured in (2) and (3) and whose combined resource usage makes up the resource usage for tees. Compositors, internally composed of decoders and mixers, were not directly prototyped or measured. While mixers are not considered when evaluating the proposed resource usage allocation heuristics (as explained in Section 3.3), decoders are an important part of it, and for them, an appropriate performance evaluation dataset was obtained.

These three evaluation workflows were orchestrated by Python[49] scripts, varying the parameters to obtain averaged data for all relevant combinations. Considering the data formats listed in Section 5.2, their combinations, and the methodology here described, we obtained measured data for 84 component variations: 64 transcoders, 8 encoders, 8 decoders, and 4 tees. Each experiment consisted of a 10-second long streaming session, repeated 25 times for each evaluated component variation, comprising a total of 2100 evaluation rounds. Results are presented next.

5.5 | Component performance evaluation dataset

Components' resource usage statistics, such as memory, CPU, bandwidth, and processing delay, were collected from these experiments. The first data entry is depicted in Table 2: the bandwidth measured for each A/V input (measured in the encoder output). All the video formats that can be used between the components are listed here.

The measured CPU usage average is shown in Figure 9. As each VM has six cores, the absolute CPU power is 600% (6*x*100%), and 100% represents the usage of one core. The CPU usage of components processing DTLS-SRTP is similar to when they process nonencrypted flows, an effect of the Advanced Encryption Standard New Operations (AES-NI),[50] an AES-dedicated instruction set embedded in Intel processors, which Gstreamer's and Linux's libraries link to. Tees use less CPU resources as they only replicate the data, without encoding/decoding the data; they show no encryption statistics because they do not handle encrypted streams themselves but always in combination with transcoders. Memory usage is shown in Figure 10 with higher resource consumption for decoding operations: We consider this to be related to the sink management exercised by the GStreamer-based decoders, a task to be handled when designing the endpoints.

Processing and starting delay statistics are depicted in Figures 11 and 12, respectively. The processing delay, the time spent between the reception of the first streaming data segments and sending the first data segments of the component's output remain low enough to make component chaining feasible (as the processing delay of a chain is the sum of all in-line components). The monitored start delay statistics (time to initialise and set-up a component) are much higher, although also low enough to keep the total delay at acceptable levels (Equation 1 details how these two delays are used

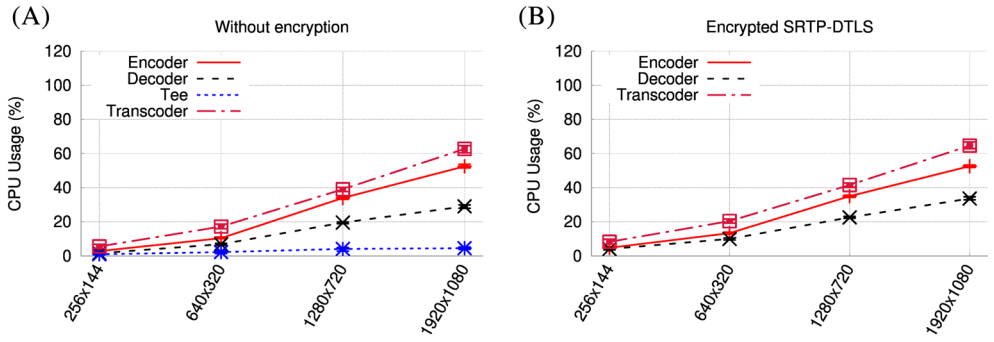

FIGURE 9 CPU usage for encoders, decoders, tees, and transcoders

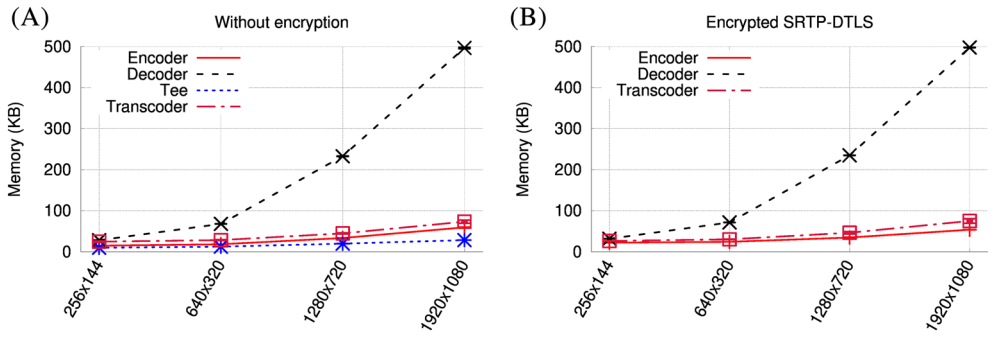

FIGURE 10 Memory usage for encoders, decoders, tees, and transcoders

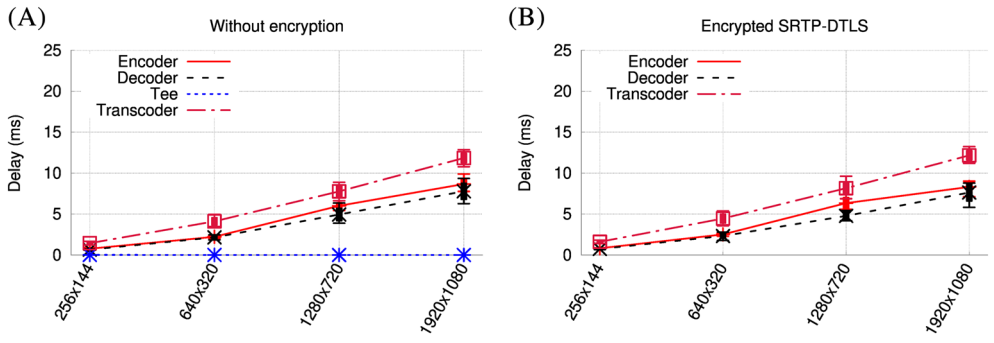

FIGURE 11 Processing delay measured for encoders, decoders, tees, and transcoders

to calculate the impact on our collaboration join time constraint of 2 seconds, detailed in Section 3.1: in the case of the start delay, required components are orchestrated/initiated in parallel; thus, the delays are not summed).These results were fed to the extended CloudSim simulator, which is used to evaluate the proposed component-to-resource allocation algorithms, of which the results are presented in the next section.

6 | SIMULATION OF COLLABORATIVE SESSIONS

This section describes the evaluation of the proposed resource allocation heuristics. A scenario with multiple teachers and students streaming course material and student feedback (ie, two-way collaboration) was modelled. The presented statistics reflect the efficiency of the component placement algorithms in face of lengthy platform usage, on top of decisions made by VM preallocation heuristics and under the requirements and deadlines depicted in Section 3.1. Evaluation results are compared with the results of a previously proposed heuristic that tackles a similar scenario, under similar project requirements.

6.1 | Simulation set-up

Our collaboration scenario generation tool was developed taking into account industrial partners' feedback.[10,37,41] In this work, it has been extended to accept encryption parameters and receive the monitored component usage statistics provided by the evaluation in Section 5. As a short imposed delay while establishing a streaming session is a strict project requirement, and the VMs that run the allocated software streaming components can take several minutes to be

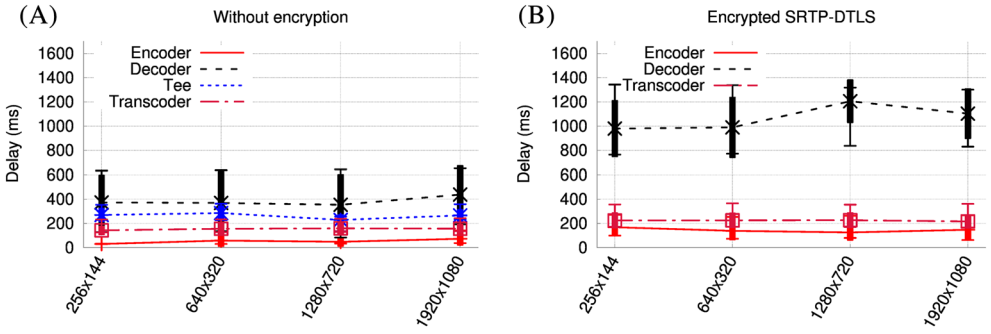

FIGURE 12 Component starting delay measured for encoders, decoders, tees, and transcoders

TABLE 3 Amazon EC2 templates used as baseline to simulate and calculate costs

Template	vCPUs	vCPU baseline	RAM (GB)	Storage	Cost/month	Guaranteed vCPUs	MIPS
t2.nano	1	5%	0.5	10GB	4.91	0.05	50
t2.micro	1	10%	1	10GB	9.81	0.1	100
t2.small	1	20%	2	10GB	19.62	0.2	200
t2.medium	2	20%	4	10GB	39.24	0.4	400
t2.large	2	30%	8	10GB	78.48	0.6	600
t2.xlarge	4	23%	16	10GB	156.95	1.0	900
t2.2xlarge	8	17%	38	10GB	313.89	1.36	1360

instantiated,[39] these VMs must be allocated beforehand. Our evaluation framework implements a previously proposed VM management approach[38] to preemptively instantiate and manage VM pools. The results presented in this article therefore do not uniquely reflect the efficiency of the proposed component placement algorithms but represent their efficiency on top of the VM preallocation decisions (ie, a two-phased approach). Therefore, instead of measuring only the cost of the VMs chosen to run the components, our cost metrics also consider preallocated VMs, which are idle while waiting for components to be deployed on them.

The utilised streaming formats are the same and are used to evaluate the components, listed in Section 5.2, and the components types are the same as depicted on Figure 3. For each component type, several combinations exist of video formats, A/V input/output count, and resolutions. The usage pattern consists of a single teacher lesson with one source location and up to 100 distributed endpoints, ie, local endpoints (teacher's classroom), remote endpoints (located at remote classrooms), mobile endpoints, and home endpoints. These endpoints are bidirectional, therefore are able to receive and provide content simultaneously (eg, student feedback).

Amazon EC2 templates[51] listed in Table 3 were used to host the components and for cost calculation. According to the Amazon EC2 design, vCPU indicates one CPU core. The vCPU baseline is a minimal guaranteed performance based on a CPU credit system,[52] with possible but not guaranteed higher bursts. Therefore, this table also shows an estimation of guaranteed cores (eg, t2.medium has 2 vCPUs and 20% as baseline, resulting in 0.4 guaranteed vCPUs available). This guarantee is an important parameter to take into account to meet near real-time demands and avoid quality of experience losses.

We consider that one vCPU core, from Amazon, Virtual Wall, or Cloudsim, provides a fixed total 1000 million instructions per second (MIPS), the processing unit used by the CloudSim simulator. This equalisation makes it feasible to compare among the Virtual Wall cores used to evaluate the prototyped components, Amazon EC2 cores used to measure costs and simulator cores used to evaluate the proposed algorithms. For instance, if a transcoder uses 50% of a CPU during the evaluation experiments, it uses 50% of a single CPU core, which we consider equivalent to 500 MIPS. This component fits a t2.large Amazon EC2 template, which costs $78.48 per month when used at full capacity. The costs are based on the calculator provided by Amazon,[53] which is also used to calculate the bandwidth cost.

Bandwidth prices are related to a monthly average usage of 1 Mbit/s: for instance, a 10 Mbit/s monthly average usage with y cost results in a monthly cost of 10y. Four different traffic types from Amazon's business model were mapped to our model:

1. Amazon's Interregion traffic, comprising the traffic among their data centres, costing $6.48. This is mapped as inter-data centre traffic in our model.
2. Amazon's Intraregion, transiting only inside a data centre at a cost of $3.24. This is mapped as intradata centre.

3. Incoming data transfer, comprised of all the data that reaches the Amazon data centre but originates from non-Amazon hosts. This is mapped as incoming traffic from the Internet, under no cost.
4. Outgoing data traffic, that considers all traffic generate by an Amazon data centre and is sent to non-Amazon hosts. Our model maps this as outgoing traffic to the Internet, which is priced as $27.81.

As the algorithms reuse resources frequently, all the monthly costs were extrapolated per second values when parameterizing the simulator. The simulator ran on an Ubuntu 14.04 VM hosted by a VMWare ESXi, with 8 virtual Intel Xeon E5-2630 CPU cores (2.3GHz) and 24GB of RAM. Metrics used to evaluate the performance of the different component-to-resource provisioning algorithms are detailed next

6.2 | Evaluation metrics

Total interruption count: average of interruptions/reestablishments per stream for the whole simulation. As clarified in Section 2, this is caused by the algorithms when adapting the stream workflow to accommodate a new content request. This metric indicates a problem only when combined with a high delay in reestablishing interrupted streams, affecting quality of experience.

Total VM cost ($): this metric summarises the costs of all VMs used during the simulation. For a simulated scenario composed of D data centres, with V_d the full set of VMs running in a data centre d, $C_{d,v}$ the cost of a VM v when running in a data centre d, and with S_v and E_v giving the time of start and shutdown of a VM v, respectively, the result is given by Equation (2).

$$\sum_{d=1}^{D}\sum_{v=1}^{V_d}(C_{d,v} \times (E_v - S_v)). \tag{2}$$

Total data centre bandwidth cost ($): sum of costs of bandwidth related to the data centres. L is the set of available data centres, $Intra_x$ the intra data centre traffic cost of data centre x, $Inter_{x,y}$ the inter data centre traffic cost between data centres x and y, and $Internet_x$ the cost of outgoing traffic from x to the Internet. So, for each network segment n from a set N_x (the set of topology network segments used by a stream workflow x), with traffic average of T_n Mbit/s, and considering S_n and D_n source and sink data centres of n, respectively, the total data centre bandwidth cost is given by Equation (3).

$$\sum_{n=1}^{N}\begin{cases} T_n \times Inter_{S_n,D_n}, & if\ (S_n = D_n) \wedge (S_n \in L)) \\ T_n \times Internet_{S_n}, & if\ ((S_n \in L) \oplus (D_n \in L)) \\ T_n \times Intra_{S_n}, & if\ ((S_n \neq D_n) \wedge (S_n \in L) \wedge (D_n \in L)) \end{cases}. \tag{3}$$

Time to start a stream (s): the time spent to start a stream, from component deployment to first transmitted data. This metric is calculated as described in Section 3, Equation (1).

Successful join rate (%): industry partners agreed a 2-second collaboration meeting join deadline as critical to maintain a good user experience,[10] and this metric represents the compliance with this agreement. Therefore, for a set S of streams, and E_s and S_s representing the respective shutdown and start time moments of a stream s, the adherence rate is provided by Equation (4).

$$\frac{1}{|S|}\sum_{s=1}^{S}\begin{cases} 1, & if\ (E_s - S_s) \leq 2s \\ 0, & if\ (E_s - S_s) > 2s \end{cases}. \tag{4}$$

Source data centre bandwidth utilisation (Mbit/s): considering a simulation with only one teacher, his location can be described as the main source data centre, and its uplink is a good spot to collect network bandwidth utilisation. The incoming and outgoing values for this bandwidth measurement spot allow the comparing of bandwidth usage characteristics of the different proposed algorithms.

Algorithm runtime (ms): once a stream is requested, the resource placement algorithms must take decisions as fast as possible to avoid negatively impacting the successful join rate. This metric indicates the algorithms performance with regards to rapid decision making. Measurement points inserted in the algorithms calls, within the source code, make this feasible.

6.3 | Simulation results

Figure 13 depicts the simulated behaviour and the consumed bandwidth: (A) the rate of streams generated by the endpoints joining and leaving the collaboration session (a data sample for the simulation of 100 endpoints) and (B) the

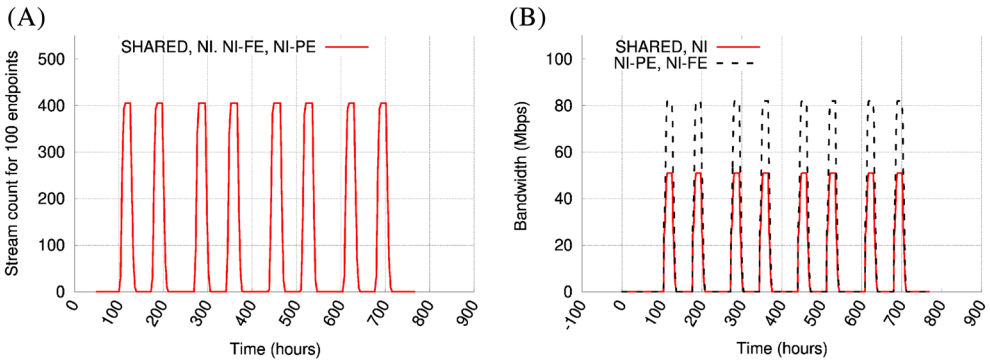

FIGURE 13 (A) The total stream count generated by 100 endpoints joining and leaving educational sessions happening twice per week. (B) The bandwidth consumed during each of these sessions, per algorithm (NI-PE and NI-FE are encrypted, SHARED, and NI are not)

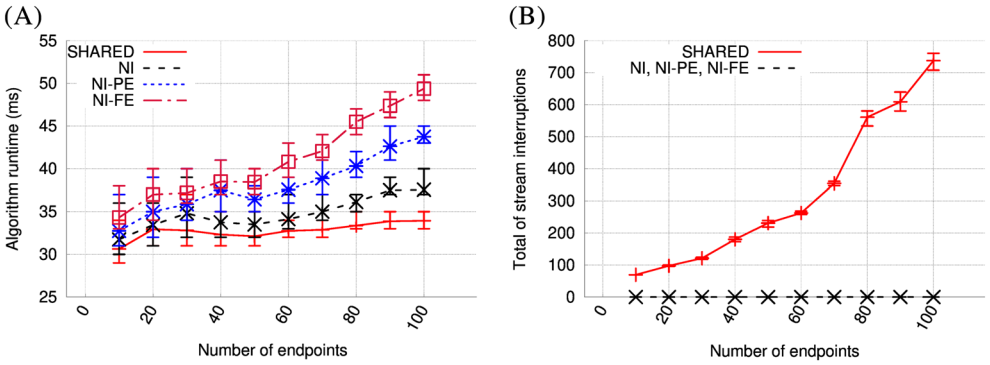

FIGURE 14 (A) The algorithms' runtime for each component chain allocation; (B) the averaged total of interruptions per stream

network traffic measured in the upstream of the teacher's location network (using the source data centre bandwidth utilisation metric), which is proportional to the amount of active streams. The stream pattern behaves according to the simulator parameterization, which mandated two active teaching sessions per week during 4 weeks. For the bandwidth, the encrypted data formats use more bandwidth, as expected. As the simulation triggers multiple streams using multiple A/V formats/resolutions, the bandwidth usage gap between encrypted and nonencrypted stream sessions, shown in Figure 13, consolidates the bandwidth usage differences shown in Table 2.

As stated before, the user experience depend in part on the time to join a collaboration. Two important measurements in this regards are processing time and number of stream interruption. The runtime is depicted in Figure 14A, with relevant but tolerable values under 51ms. The interruption, depicted in Figure 14B, shows a high number of stream interruptions for the SHARED algorithm, while the newly proposed ones, NI, NI-PE, and NI-FE, as expected, do not interrupt the existing collaboration streams. The steep interruptions growth for SHARED shows up because all streams interrupted are counted: for instance, if a tee is replicating a stream to 10 decoders and had its input interrupted, the 10 streams allocated on those decoders will be taken into account. As the components experimentation presented in Section 5 showed high delays to (re-)establish streams, interruption-free algorithms like NI, NI-PE, and NI-FE are clearly favoured.

The proposed heuristics that avoid stream interruptions preserve the user experience but require more resources. Figure 15 shows the amount of components allocated by each proposed algorithm during the whole simulation. Figure 15A shows the tee count and depicts the effort made to avoid stream interruptions, allocating several more tees for NI, NI-HE, and NI-FE, in comparison with SHARED. Affected by the enforcement of encryption, the total of placed transcoders is shown in Figure 15B: NI-FE uses more transcoders to guarantee that the data are always encrypted. Finally, Figure 15C shows the amount of allocated encoders, decoders, and compositors, numbers that stay the same for all algorithms.

Figure 16 shows the costs for (A) VMs and (B) bandwidth. SHARED presents a lower total VM cost, but unacceptable stream interruptions (as shown in Figure 2). The other algorithms solve the interruption problem at a higher cost than SHARED, increasing the VM cost by up to 10.3%, mostly related to the preinstantiation of additional tee components to avoid stream interruptions. Among NI, NI-PE, and NI-FE, the costs increase with the level of encryption enforcement: while NI uses no encryption, maintaining a lower cost, NI-PE deploys encryption partially, increasing the costs because of the use of additional transcoders. NI-FE uses more transcoders to fully encrypt the stream workflow, raising the costs by up to 23.3% when compared with SHARED, but within reasonable limits. With regards to the bandwidth, algorithms provisioning secure components increase the costs by up to 48.38%, as encrypted data formats use more bandwidth (detailed in Table 2).

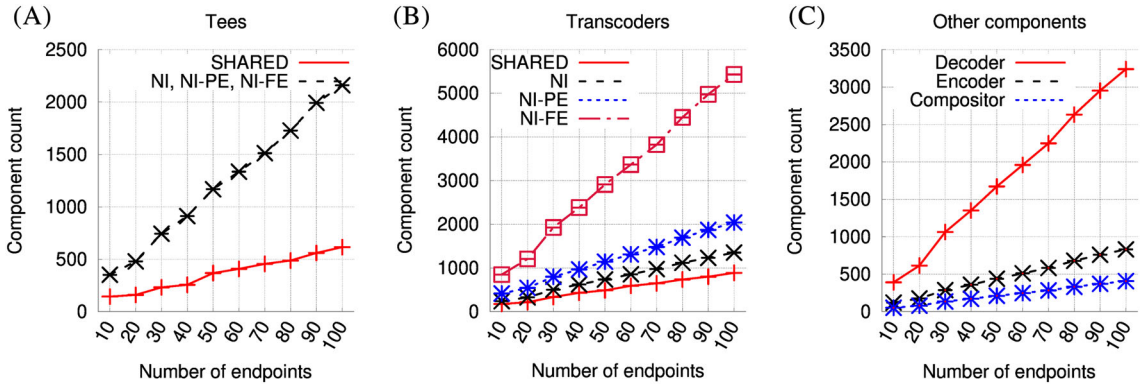

FIGURE 15 (A,B) The total of tees and transcoders allocated during the simulation of each of the proposed component allocation heuristics; (C) the total of allocated decoders, encoders, and compositor (same quantities of these are allocated by all proposed algorithms)

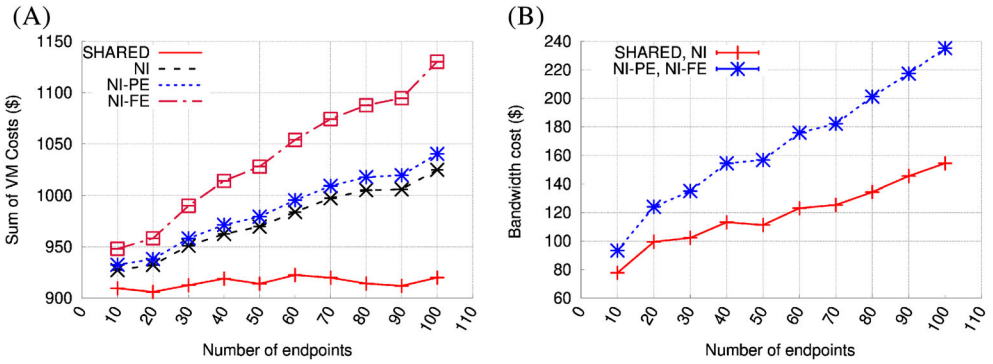

FIGURE 16 Total cost of VMs and bandwidth, both accounted for the whole simulation

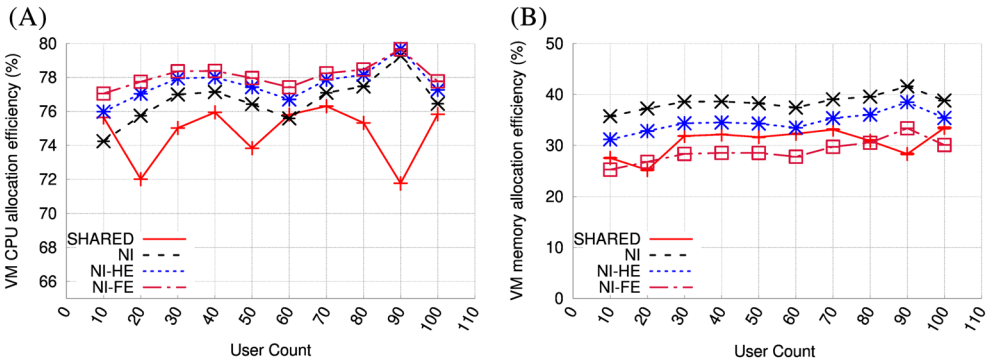

FIGURE 17 VM template's capabilities not always match component's requirements, and resources can be wasted. This graph shows the allocation efficiency: (A) the averaged percentage of CPU usage among all VMs that run components; (B) the same information for memory usage

The VM templates (listed in Table 3) offer capacities that differ from the components' requirements (presented in Section 5.5), and placement decisions may result in waste of resources. The component allocation algorithms' efficiency w.r.t. nonutilised CPU and memory are depicted in Figure 17A,B, respectively. The peaks seen in the graphs are expected because distinct component placement algorithms and user demand levels are evaluated, and these influence the amount of VMs from each template the VM allocation algorithms preemptively allocate. These efficiency values result from a trade-off between the set of VM templates' size and VM costs: while having more varied templates could better fit the components and waste less resources, it would increase the operational costs imposed by preemptively allocating more VMs. A proposed future work is targeted at improving this by sharing VMs among multiple containerised components.

The efficiency of the VM allocation algorithms was also measured. They preemptively allocate VMs, and to guarantee that the demand is met in time, more VMs than necessary can be instantiated. The percentage of VMs utilised to host components is shown in Figure 18A. A lower demand affects more the efficiency (varying from 56% to 85% for 10 concurrent users), and when more demanded (for 100 concurrent users), it scales to values from 85% to 97%. The VM allocation algorithms are an important support towards the compliance with the 2-second deadline imposed to the component allocation

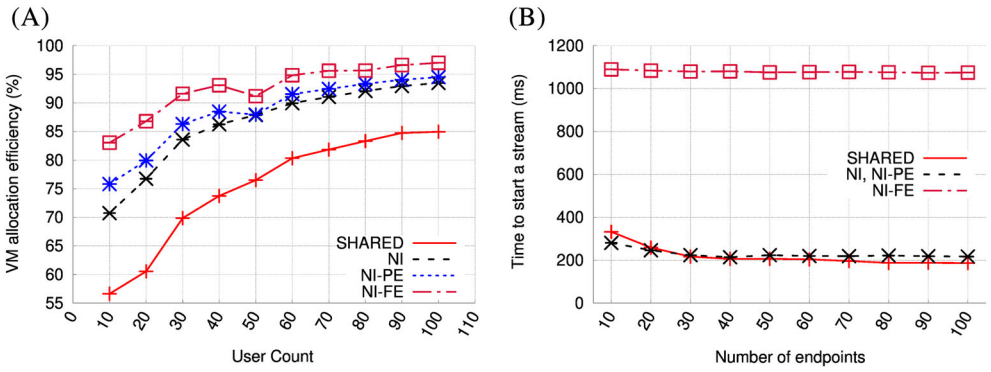

FIGURE 18 (A) The percentage of instantiated VMs used to host components, in contrast with VMs instantiated but not used; (B) the time spent to place a stream in the stream workflow, from the stream request generated by the endpoint to the reception of the streamed content

heuristics, and inefficient behaviour increases costs. Therefore, this efficiency level must be accepted as a trade-off between cost and project requirements.

Finally, Figure 18B displays the adherence to the A/V collaboration join limit of 2 seconds, when an endpoint joins an A/V collaboration session. The average time remains under 1.2 seconds and is slightly reduced upon increase in the number of active users. Reason for this is the reutilisation of tees: The components placed before the tee are already running and do not need to be initialised. Comparing algorithms that provision secured (encryption-enabled) and nonsecured components, the utilisation of encrypted decoders affects significantly the time to establish a session (the decoder loading time is shown in Figure 12). While for SHARED, NI, and NI-PE, which use nonencrypted decoders, the average join time stays between 0.2 and 0.37 seconds, NI-FE, that enforces encrypted decoders, raise it to 1.1 seconds, a value compatible with the 2-second deadline described in Section 3.

These collaboration simulation results, combined with the components evaluation from Section 5, show that SHARED is not a good option because of stream interruptions. This is handled by the nonencrypted NI algorithm. On the other hand, the fully encrypted version NI-FE presents a solution with an acceptable protection-cost trade-off. Additionally, if the risk of no encryption inside the data centres can be taken to acceptable levels by using other security countermeasures, NI-PE can encrypt only the external traffic providing a lower cost entry point than NI-FE, a choice which must be made based on personalised risk assessment.

7 | CONCLUSION AND FUTURE WORK

In this article, we have presented cloud provisioning algorithms and models tuned for encryption-enabled professional collaborative media applications and presented evaluation results of the proposed component-to-cloud resource allocation algorithms applied to a distributed educational collaboration scenario. The CloudSim simulation framework has been extended to implement this collaborative media application model and implemented the collaborative media application model. These needs originated from a business migration from a static hardware-based collaboration platform to an elastic cloud-hosted model protected by policy-based security controls.[9]

No dataset was available to feed the simulator with encryption-enabled streaming components' resource usage statistics. Therefore, components such as encoders, decoders, transcoders, and tees, all described in Section 3, were prototyped in encryption-enabled versions and evaluations gathered usage statistics. These statistics showed acceptable resource usage levels, but component initialisation delay values which were bigger than expected and harmful to the user experience. For instance, when a new endpoint joins the platform and requests to join an existing collaboration stream, the current flow must be interrupted to accommodate a new in-line positioned tee, causing the reestablishment of the stream after the tee initialisation. This can cause unacceptable interruptions for active users, and new interruption-free algorithms were proposed to handle this concern.

Simulation results show that a previous algorithm, named SHARED, can harm the user experience by unacceptably interrupting live data streams. The proposed NI, NI-HE, and NI-FE solve the interruption issue and can also handle encrypted component allocation decisions. While NI has encryption disabled, NI-HE partially encrypts, focusing on infrastructure spots more subject to security threats, and NI-FE fully encrypts the VM traffic, all three under acceptable costs. Therefore, a business decision must be made in order to assess the security risks and choose one of them as the best cost-benefit solution.

The research carried out raised possible future work avenues. Although having achieved linear and acceptable costs, we believe that the difference among encrypted and nonencrypted options can be minimised to a level that makes it feasible to enforce encryption on any situation. Dynamic tees able to distribute DTLS-SRTP streams with distinct video formats among multiple sinks (and without auxiliary transcoders) can reduce costs of the encrypted approach, a feasibility to be investigated. Architecturewise, a more decentralised microservices-based approach can run the components allocation algorithms and send component initialisation commands. This can reduce the component instantiation delay while attaining higher scalability and resilience levels. Mathematical modelling of the problem allows to compare the proposed algorithms with the optimal case in terms of costs and delay. The scenario proposed in this article, with component placement techniques acting on top of the VM pre-allocation heuristics, is complex. However, we believe that integer linear programming (ILP) modelling of it in two separate parts, one focused on VM preallocation and the other on component placement, will allow us to evaluate the optimality of our heuristic solutions. Finally, it is know that VMs instantiation delays can last several seconds (between 44 and 810 s depending on the operational system and the size of the VM).[39] Multiple software components hosted by a single server, using specialised software components or containers,[54] can reduce instantiation delay and the need for a larger and costly pool of initialised VMs.

ACKNOWLEDGEMENTS

The research described in this article is partially funded by the imec EMD ICON research project and the FWO projects G025615N "Optimised source coding for multiple terminals in self-organising networks" and G059615N "Service-oriented management of a vitalised future internet."

ORCID

Rafael Xavier https://orcid.org/0000-0001-6622-1357

REFERENCES

1. WhatsApp. WhatsApp Messenger and Collaboration Tool. https://www.whatsapp.com. Accessed on April 10, 2018.
2. Facebook. Facebook Messenger. https://www.messenger.com/. Accessed on April 10, 2018.
3. Google. Google Hangout. https://hangouts.google.com/. Accessed on April 10, 2018.
4. Microsoft. Enterprise capabilities with Skype for Business in Office 365. http://products.office.com/en-us/skype-for-business/online-meetings. Accessed on April 10, 2018.
5. Cisco. Cisco Hosted Collaboration Solution (HCS). http://www.cisco.com/web/solutions/hcs/index.html. Accessed on April 10, 2018.
6. Barco NV. Solutions for higher education and training. https://www.barco.com/en/Products/Collaborative-Learning/Solutions-for-higher-education-and-training. Accessed on April 10, 2018.
7. Courcoubetis C, Dramitinos M, Stamoulis GD, Blocq G, Miron A, Orda A. Inter-carrier interconnection services: Qos, economics and business issues. In: 2011 IEEE Symposium on Computers and Communications (ISCC). Kerkyra, Greece: IEEE; 2011:779-784.
8. Microsoft. Media Quality and Network Connectivity Performance in Skype for Business Online. https://docs.microsoft.com/en-us/skypeforbusiness/optimizing-your-network/media-quality-and-network-connectivity-performance. Accessed on February 22, 2019.
9. Institute Imec Research. EMD ICON Project: Elastic Media Distribution for Online Collaboration. https://www.imec-int.com/en/what-we-offer/research-portfolio/emd. Accessed on April 10, 2018.
10. Xavier R, Granville LZ, Volckaert B, De Turck F. Elastic resource allocation algorithms for collaboration applications. *J Netw Syst Manag*. 2017;25(4):699-734.
11. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice Experience*. 2011;41(1):23-50.
12. Imec Research Institute. jFed Framework. https://jfed.ilabt.imec.be. Accessed on April 10, 2018.
13. GStreamer. GStreamer: open source multimedia framework. https://gstreamer.freedesktop.org. Accessed on April 10, 2018.
14. Szewczyk P, Macdonald R. Broadband router security: History, challenges and future implications. *J Digit Forensic Secur Law*. 2017;12(4):6.
15. Sermpezis P, Kotronis V, Dainotti A, Dimitropoulos X. A survey among network operators on BGP prefix hijacking. *ACM SIGCOMM Comput Commun Rev*. 2018;48(1):64-69.
16. Cheng N, Wang XO, Cheng W, Mohapatra P, Seneviratne A. Characterizing privacy leakage of public WiFi networks for users on travel. In: 2013 Proceedings IEEE INFOCOM. Turin, Italy: IEEE; 2013:2769-2777.
17. Dilkash Neha, Gupta Anku, Jain Arpita. Real time video encryption for secure multimedia transfer: a novel approach. *Int J Eng Sci*. 2018:17077.
18. Sombatruang N, Kadobayashi Y, Sasse MA, Baddeley M, Miyamoto D. The continued risks of unsecured public Wi-Fi and why users keep using it: Evidence from Japan. In: 2018 16th Annual Conference on Privacy, Security and Trust (PST). Belfast, UK: IEEE; 2018:1-11.

19. Abolghasemi MS, Sefidab MM, Atani RE. Using location based encryption to improve the security of data access in cloud computing. In: 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Mysore, India: IEEE; 2013:261-265.
20. Bhatti R, Damiani ML, Bettis DW, Bertino E. Policy mapper: Administering location-based access-control policies. *IEEE Internet Comput*. 2008;12(2):38-45.
21. Karimi R, Kalantari M. Enhancing security and confidentiality on mobile devices by location-based data encryption. In: 2011 17th IEEE International Conference on Networks. Singapore, Singapore: IEEE; 2011:241-245.
22. Everett C. A risky business: Iso 31000 and 27005 unwrapped. *Comput Fraud Secur*. 2011;2011(2):5-7.
23. Kothmayr T, Schmitt C, Hu W, Brünig M, Carle G. A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication. In: 37th Annual IEEE Conference on Local Computer Networks - Workshops. Clearwater, FL, USA: IEEE; 2012:956-963.
24. Bergkvist A, Burnett DC, Jennings C, Narayanan A, Aboba B. Webrtc 1.0: Real-time communication between browsers. *Working draft*. 2012;W3C:91.
25. Sivakorn S, Keromytis AD, Polakis J. That's the way the cookie crumbles: evaluating HTTPS enforcing mechanisms. In: Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society. Vienna, Austria: ACM; 2016:71-81.
26. Al Fardan NJ, Paterson KG. Lucky thirteen: Breaking the TLS and DTLS record protocols. In: 2013 IEEE Symposium on Security and Privacy. Berkeley, CA, USA: IEEE; 2013:526-540.
27. Jennings B, Stadler R. Resource management in clouds: survey and research challenges. *J Netw Syst Manag*. 2015;23(3):567-619.
28. Koslovski G, Soudan S, Gonçalves P, Vicat-Blanc P. Locating virtual infrastructures: Users and INP perspectives. In: 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops. Dublin, Ireland: IEEE; 2011:153-160.
29. Alicherry M, Lakshman TV. Network aware resource allocation in distributed clouds. In: 2012 Proceedings IEEE INFOCOM; 2012; Orlando, FL, USA:963-971.
30. Steiner M, Gaglianello BG, Gurbani V, et al. Network-aware service placement in a distributed cloud environment. *SIGCOMM Comput Commun Rev*. 2012;42(4):73-74.
31. Zhu Y, Liang Y, Zhang Q, Wang X, Palacharla P, Sekiya M. Reliable resource allocation for optically interconnected distributed clouds. In: 2014 IEEE International Conference on Communications (ICC); 2014; Sydney, NSW, IEEE:3301-3306.
32. Group ETSI Industry. Network Function Virtualisation NFV. http://www.etsi.org/technologies-clusters/technologies/nfv. Accessed on April 10, 2018.
33. Clayman S, Maini E, Galis A, Manzalini A, Mazzocca N. The dynamic placement of virtual network functions. In: 2014 IEEE Network Operations and Management Symposium (NOMS). Krakow, Poland: IEEE; 2014:1-9.
34. Moens H, De Turck F. VNF-P : a model for efficient placement of virtualized network functions. In: Proceedings of the 10th International Conference on Network and Service Management (CNSM 2014); 2014; Rio de Janeiro, Brazil:418-423.
35. Soltanian A, Naboulsi D, Salahuddin MA, Glitho R, Elbiaze H, Wette C. Ads: Adaptive and dynamic scaling mechanism for multimedia conferencing services in the cloud. In: 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC). Las Vegas, NV, USA: IEEE; 2018:1-6.
36. Chunlin L, Chuanli M, Yi C, Youlong L. Optimal media service selection scheme for mobile users in mobile cloud. *Wireless Netw*. 2019; 25(6):3179-3192.
37. Xavier R, Moens H, Volckaert B, De Turck F. Design and evaluation of elastic media resource allocation algorithms using CloudSim extensions. In: 2015 11th International Conference on Network and Service Management (CNSM); 2015; Barcelona, Spain:318-326.
38. Xavier R, Moens H, Volckaert B, De Turck F. Adaptive virtual machine allocation algorithms for cloud-hosted elastic media services. In: 2016 IEEE/IFIP Network Operations and Management Symposium (NOMS). Istanbul, Turkey: IEEE; 2016:564-570.
39. Mao M, Humphrey M. A performance study on the VM startup time in the cloud. In: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD). Honolulu, HI, USA: IEEE; 2012:423-430.
40. Xavier R, Moens H, Volckaert B, De Turck F. Resource allocation algorithms for multicast streaming in elastic cloud-based media collaboration services. In: 2016 IEEE 9th International Conference on Cloud Computing (CLOUD). San Francisco, CA, USA: IEEE; 2016:947-950.
41. Xavier R, Moens H, Slowack J, et al. Cloud resource allocation algorithms for elastic media collaboration flows. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). Luxembourg City, Luxembourg: IEEE; 2016:440-447.
42. Internet Engineering Task Force (IETF). Options for Securing RTP Sessions. https://tools.ietf.org/html/rfc7201. Accessed on April 10, 2018.
43. De Groef W, Subramanian D, Johns M, Piessens F, Desmet L. Ensuring endpoint authenticity in WebRTC peer-to-peer communication. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. Pisa, Italy; 2016:2103-2110.
44. Ng K-F, Ching M-Y, Liu Y, Cai T, Li L, Chou W. A P2P-MCU approach to multi-party video conference with WebRTC. *Int J Future Comput Commun*. 2014;3(5):319.
45. ISO I, Std I. Iso 27005: 2011. Information technology-security techniques-information security risk management. ISO; 2011.
46. Elliott C, Falk A. An update on the geni project. *ACM SIGCOMM Computer Communication Review*. 2009;39(3):28-34.
47. Vermeulen B, Van de Meerssche W, Walcarius T. jfed toolkit, fed4fire, federation. http://groups.geni.net/geni/raw-attachment/wiki/GEC19Agenda/DeveloperTopics/GEC19_jfed_bvermeul.pdf. Accessed on on June 24, 2019.
48. Xavier R. Prototyped streaming components. http://users.ugent.be/~jxavierd/EMD. Accessed on April 10, 2018.

49. Van Rossum G, Drake FL. *The Python Language Reference Manual*. Network Theory Ltd. ISBN:1906966141 9781906966140. 2011.
50. Gueron S. Intel's new AES instructions for enhanced performance and security. In: Dunkelman O, ed. *Fast Software Encryption*, Vol. 5665. Berlin, Heidelberg: Springer; 2009:51-66.
51. Amazon Inc. Amazon Elastic Compute Cloud (EC2) images. https://aws.amazon.com/pt/ec2/instance-types/. Accessed on April 10, 2018.
52. Amazon Inc. AWS CPU Credits and Baseline Performance. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-credits-baseline-concepts.html. Accessed on April 10, 2018.
53. Amazon Inc. AWS Simple Monthly Calculator. https://calculator.s3.amazonaws.com/index.html. Accessed on April 10, 2018.
54. Soltesz S, Pötzl H, Fiuczynski ME, Bavier A, Peterson L. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review*. 2007;3.41:275-287.

AUTHOR BIOGRAPHIES

Rafael Xavier is a PhD student at the Department of Information Technology of the Ghent University, imec, Belgium. He received his MSc degree from the Institute of Informatics of the Federal University of Rio Grande do Sul, Brazil. His research interests include network management, software-defined networking, network functions virtualisation, security, and optimisation of cloud-based applications.

Lisandro Zambenedetti Granville is a professor at the Institute of Informatics of the Federal University of Rio Grande do Sul, Brazil. He is co-chair of the Network Management Research Group (NMRG) of the IRTF and president of the Brazilian Computer Society (SBC). His topics of interest include network management, software-defined networking, and network functions virtualisation.

Filip De Turck is a professor at the Department of Information Technology of the Ghent University, imec, Belgium. His research interests include telecommunication network and service management and design of efficient virtualised network systems. He serves as Chair of the IEEE Technical Committee on Network Operations and Management (CNOM) and is in the Editorial Board of several journals on network and service management.

Bruno Volckaert is a professor of advanced programming and software engineering in the Department of Information Technology (INTEC) at Ghent University and senior researcher at imec. He obtained his Master of Computer Science degree in 2001, after which he investigated network aware grid service management in his PhD. His current research deals with reliable and high performance distributed software systems for a.o. City-of-Things, UAVs, intelligent railway applications and autonomous optimisation of cloud-based applications.

How to cite this article: Xavier R, Granville LZ, De Turck F, Volckaert B. Efficient allocation of elastic inter-federation encrypted streaming sessions. *Int J Network Mgmt*. 2019;29:e2079. https://doi.org/10.1002/nem.2079