École Normale Supérieure de Lyon

Rapport de stage de fin de Licence 3

Roméo La Spina

Etude de la scalabilité des serveurs de visio-conférence

Tuteur : Alain Tchana
Encadrant : Gilles Muller, INRIA Rennes

1er juin 2020 — 31 juillet 2020

Table des matières

Remerciements

Je tiens à remercier toute l'équipe Whisper de l'INRIA Paris qui m'a suivi durant le stage ainsi que dans la rédaction de ce rapport.

Tout d'abord, je remercie Gilles Muller, mon maître de stage, qui a su me guider dans cette étude et dont les idées m'ont permis d'orienter ma recherche afin d'obtenir des résultats et de bonnes connaissances sur le sujet.

Je tiens ensuite à remercier mon tuteur, Alain Tchana, qui a su me conseiller lors de la recherche de mon sujet de stage de Licence 3 et qui m'a tourné vers ce stage.

Je remercie enfin David Bromberg, membre de l'équipe Wide à l'INRIA Rennes, qui m'a donné de très bons conseils pour mettre en place les outils nécessaires pour aboutir à des résultats concluants.

1 Contexte et introduction

Les mesures de confinement appliquées dans le monde entier lors de la crise sanitaire liée à la pandémie de Covid-19 ont eu pour conséquence l'utilisation massive des systèmes de vidéo-conférence.

La robustesse, en terme de capacité à passer à l'échelle, de ces systèmes a été mise à rude épreuve. La scalabilité, c'est à dire la capacité d'un système à conserver ses performances lors de son utilisation à très grande échelle, est donc dans ce contexte un critère majeur d'efficacité d'une plateforme de visio-conférence.

Pour faire face à cette situation d'urgence, la solution adoptée a été le surdimensionnement des ressources allouées à ces systèmes. C'est notamment le cas avec le fournisseur de cloud ensemblescaleway qui a mis sur pied une version gonflée en ressources du système Zoom, afin d'éviter la saturation de ce dernier.

Cette approche naïve est compréhensible dans une situation d'urgence. Cependant, elle ne peut pas être envisagée à long terme, compte tenu du fait que la fin de la pandémie n'est pas encore en vue. C'est pourquoi il est nécessaire d'améliorer drastiquement la scalabilité de ces systèmes.

Très peu de travaux de recherche ont auparavant étudié l'efficacité de ces systèmes de visio-conférence. L'objectif de notre travail dans le cadre de ce stage était de comprendre les besoins en ressources de ces systèmes, à petite et à grande échelle. Le but ultime de cette étude est d'améliorer l'implantation de ces systèmes afin qu'ils utilisent efficacement les ressources, et afin de minimiser l'impact énergétique d'une utilisation intensive des plateformes de visio-conférence.

Dans le cadre de ce stage, nous avons utilisé la plateforme de visio-conférence Jitsi Meet. La plateforme Jitsi Meet est une alternative open source à des systèmes similaires comme Zoom, et offre notamment la possibilité pour les utilisateurs, en plus de l'audio et de la vidéo, d'effectuer un partage d'écran ainsi que de s'échanger des messages texte (chat).

Ce système fonctionne suivant un modèle client-serveur : l'application Jitsi Meet est hébergée sur un serveur, et les participants (les clients) d'une conférence vont se connecter à ce même serveur, qui servira pour faire transiter les flux et les connexions d'un participant aux autres.

L'enjeu de l'étude sera de tester l'application en faisant des tests à grande échelle, tout en monitorant l'utilisation des ressources côté client et côté serveur, afin de connaître les limites de la scalabilité du système.

Nous allons dans un premier temps présenter le fonctionnement de la plateforme Jitsi Meet, les modes de communication entre les différents composants du système, puis nous verrons comment nous avons déployé notre installation côté client et côté serveur pour pouvoir réaliser des tests de scalabilité, et enfin nous présenterons les résultats obtenus à l'issue de ces tests.

2 Présentation de la plateforme Jitsi Meet

2.1 Présentation des différents modes de distribution des flux multimédia

Il existe trois principales méthodes pour distribuer les flux audio et vidéo entre les participants d'une session de visio-conférence : Mesh, SFU et MCU [2]. La figure 1 illustre leur fonctionnement et le tableau en figure 2 résume leurs différences.

2.1.1 Mode Mesh ou P2P (peer-to-peer)

La méthode Mesh consiste à faire communiquer les participants directement entre eux, sans passer par un serveur.

L'avantage d'une telle méthode est qu'il n'y a pas besoin de serveur pour héberger la plateforme, et par conséquent la connexion est plus rapide. Le principal problème de ce mode est que pour une conférence à n participants, chaque participant envoie $n-1$ flux et en reçoit $n-1$, or la bande passante montante est souvent beaucoup moins élevée que la bande passante descendante notamment dans le cas de l'ADSL. Ainsi, la bande passante des participants sera rapidement saturée pour un nombre de participants supérieur ou égal à 5. Toutefois, Mesh reste un bon mode de distribution pour des petites sessions (au plus quatre participants).

2.1.2 SFU (Selective Forwarding Unit)

Le mode SFU consiste à centraliser tous les flux multimédia envoyés par les participants vers un serveur, qui va ensuite tous les redistribuer à chacun des participants.

Avec cette méthode, la bande passante montante nécessaire est considérablement réduite par rapport au mode Mesh car les participants n'ont qu'un flux à envoyer au serveur.

Toutefois, cela nécessite que le serveur ait une excellente bande passante montante étant donné qu'il doit envoyer $n-1$ flux à chaque participant, donc $n(n-1) = \Theta(n^2)$ flux au total. En pratique, ce n'est pas très contraignant car un serveur peut facilement avoir une bande passante sortante de l'ordre du Go/s dans une structure de type data center.

2.1.3 MCU (Multipoint Control Unit)

La méthode MCU est une méthode beaucoup plus complexe qui consiste à mixer au niveau du serveur l'ensemble des flux des participants pour n'avoir à en renvoyer qu'un seul à chaque participant. Avec un tel mode, chaque participant n'envoie et ne reçoit qu'un seul flux vidéo.

Il semblerait alors que ce soit la meilleure méthode, toutefois elle pose un problème de taille : les opérations de mixage/encodage au niveau du serveur ainsi que l'opération de décodage au niveau du client sont extrêmement coûteuses en capacité de calcul. C'est pourquoi les solutions de type MCU restent pour l'heure très peu utilisées du grand public, même si elles ont connu un important développement ces dernières années.

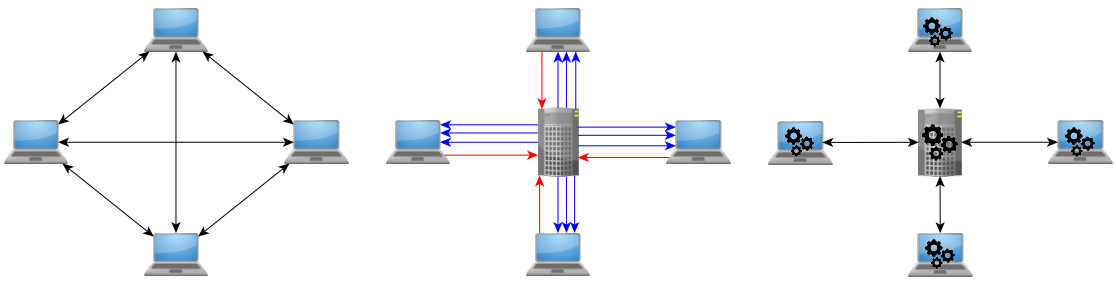


FIGURE 1 – De gauche à droite, les modes Mesh, SFU et MCU

	Mesh/P2P	SFU	MCU
Flux client envoyés	$n-1$	1	1
Flux client reçus	$n-1$	$n-1$	1
Flux serveur envoyés	-	$n(n-1)$	n
Flux serveur reçus	-	n	n
Remarques	bien pour de petites sessions, pas besoin de serveur	nécessite une bonne bande passante pour le serveur	coûteux en calcul

FIGURE 2 – Comparaison des trois modes

2.2 Fonctionnement de l'API WebRTC

WebRTC (Web Real Time Communication) est une interface de programmation (API), open source, écrite en JavaScript et supportée nativement par la plupart des navigateurs, qui permet d'échanger des flux de données (notamment audio et vidéo) entre des utilisateurs (par exemple des participants d'une session de visio-conférence)

Le principe de l'API WebRTC consiste à passer par un serveur pour établir la communication entre deux utilisateurs, qui pourront alors communiquer des flux multimédia directement. Ainsi, pour qu'Alice puisse communiquer avec Bob, il suffit qu'Alice et Bob soient tous les deux connectés au serveur qui héberge l'application WebRTC (par exemple le serveur qui héberge Jitsi Meet). Lorsque Alice fait une requête pour communiquer avec Bob,

— Le serveur relaie la demande d'Alice à Bob
— Si Bob accepte la demande, l'information est transmise à Alice
— Le serveur crée deux objets de type PeerConnection qui permettent à Alice et Bob de communiquer dans les deux sens.

L'avantage de cette API est qu'elle fonctionne indépendamment de toute extension pour les navigateurs. Elle est donc très simple à utiliser.

2.3 Description du protocole XMPP

La plateforme Jitsi Meet utilise, dans le cadre de son fonctionnement interne que l'on verra en section 2.4, le protocole XMPP (Extensible Messaging and Presence Protocol). Ce protocole est très utilisé dans les services de messagerie instantanée. Il est dérivé du projet open source Jabber, et est basé sur la transmission de données au format XML.

XMPP fonctionne suivant un modèle client-serveur. Lors de l'établissement d'une connexion XMPP, deux flux de données XML sont créés (client vers serveur et serveur vers client). Les "paquets" de données XML échangées entre le client et le serveur sont appelées des "stanzas" XMPP.

La particularité du protocole XMPP est que le serveur est à tout moment capable de savoir si le client est toujours connecté. Tant que c'est le cas, alors il envoie directement au client tous les messages qui lui sont destinés. En revanche, lorsque le client est déconnecté, le serveur stocke tous les messages destinés au client, et dès la prochaine connexion, le client pourra recevoir tous ces messages. C'est donc un excellent protocole pour les services de messagerie.

BOSH (Bidirectional-streams Over Synchronous HTTP) est une technique permettant d'établir une connection HTTP bidirectionnelle [5]. Le principe de cette technique est le sui-

vant : Alice envoie une requête HTTP à Bob, qui reste en attente tant qu'il n'a pas d'information à envoyer à Alice. Lorsque Bob envoie une information à Alice, cette dernière renvoie une requête afin de maintenir la connexion ouverte (ce principe est appelé *long polling*).

Cette technique est une des meilleures solutions pour transporter des données XML de manière bidirectionnelle avec le protocole HTTP. En particulier, elle peut s'appliquer pour transmettre des stanzas XMPP à l'aide de requêtes HTTP. On peut aussi utiliser cette technique pour d'autres protocoles basés sur des données XML. Ainsi, Alice et Bob pourront simplement s'échanger les stanzas XMPP sur le port 80/TCP (HTTP), plutôt que sur le port 5222 réservé aux connexions XMPP.

2.4 Fonctionnement global de Jitsi Meet

La plateforme de visio-conférence Jitsi Meet utilise le mode Mesh pour des conférences à deux participants, puis SFU pour des conférences à trois participants ou plus. Les connexions entre les participants, ou entre les participants et le serveur, sont établies grâce au protocole de l'API WebRTC.

L'installation repose sur les composants suivants (voir figure 3) :

— Un serveur web, par exemple `nginx`, configuré de manière à ce que l'URL du serveur (`scalevision.irisa.fr`) pointe sur les pages Javascript de la plateforme Jitsi Meet

— `jitsi-videobridge` qui sert d'intermédiaire pour distribuer les flux audio et vidéo entre les participants d'une conférence selon le mode SFU. Pour des sessions de 2 personnes, les flux sont transmis en peer-to-peer et `jitsi-videobridge` n'est pas utilisé. Sur l'application Jitsi Meet, tous les flux vidéo sont encodés à l'aide du codec VP8.

— `jicofo` (JItsi COnference FOcus) qui gère les interactions entre les participants, la création de nouvelles sessions, l'intégration d'un nouveau participant à une conférence existante, etc...

— Le serveur de communication XMPP `prosody`.

Lorsqu'un client se connecte à la plateforme, il fait d'abord une requête SSL sur le port 443 pour obtenir l'accès. Le client arrive sur la page d'accueil de la plateforme. Supposons qu'il veuille créer ou rejoindre une conférence en entrant sur la page d'accueil le nom de la conférence. L'information concernant cette nouvelle connexion est reçue par la plateforme Jitsi-Meet, qui est ensuite communiquée à `jicofo` par l'intermédiaire du serveur `nginx` et du serveur `prosody`. Si la conférence demandée par le client existait déjà, `jicofo` ajoutera ce nouveau participant à la conférence. Sinon, `jicofo` créera une nouvelle conférence.

Une fois que le client est affecté à une conférence, les flux vidéo et audio qu'il fournit (par exemple, grâce à sa webcam et son microphone) passent par le port UDP 10000 qui est écouté par `jitsi-videobridge`. Enfin, `prosody` est un serveur XMPP qui sert à rediriger les flux multimédia couplés aux signaux envoyés sous forme de stanzas XMPP par `jicofo` (création d'une nouvelle conférence, changement du participant qui est en train de parler (le focus), ...) vers la plateforme JavaScript de Jitsi Meet. `prosody` reçoit ces informations sur le port 5347 puis les transfère sur le port 5280 en suivant le protocole XMPP sur BOSH. Ce port est finalement écouté par `nginx` qui est configuré en mode proxy (voir figure 4) pour pouvoir relayer les informations et les flux multimédia à la plateforme Jitsi Meet.

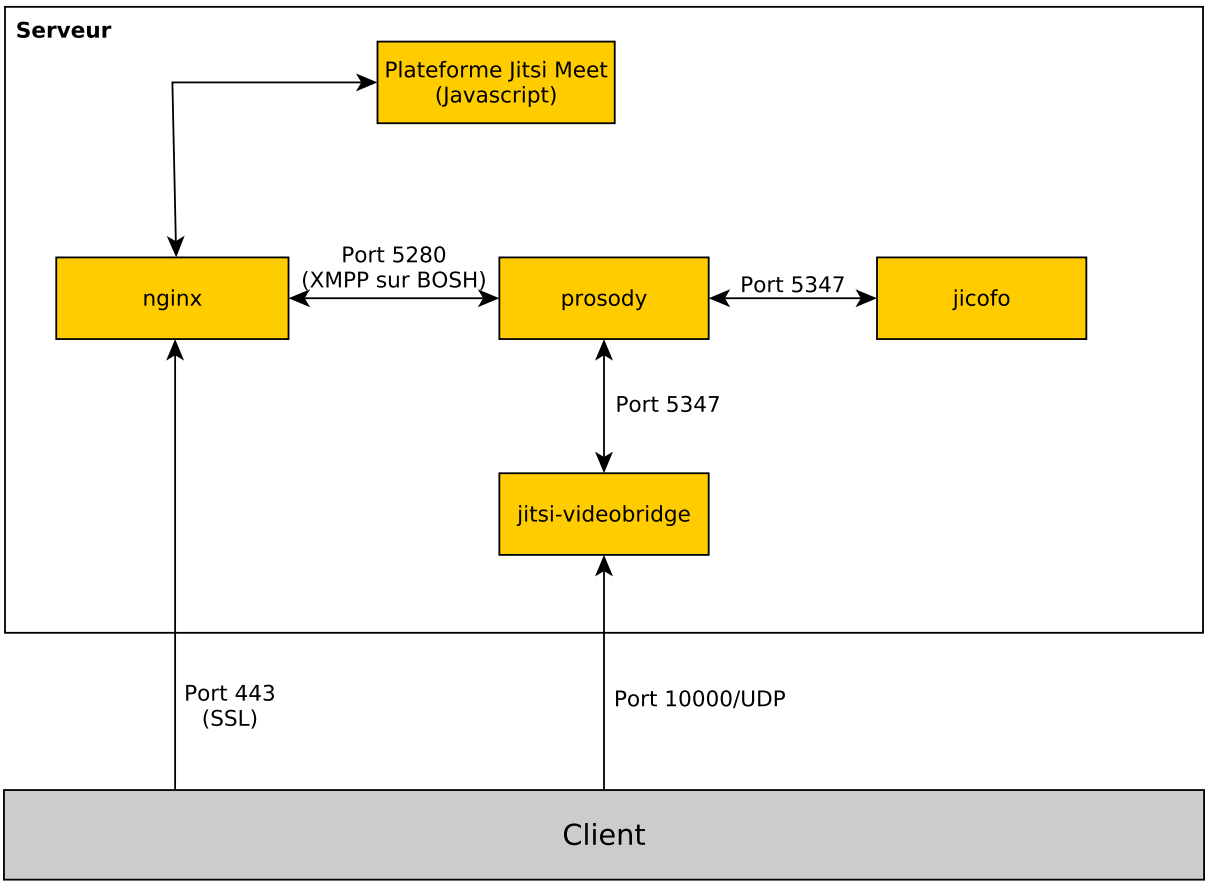


FIGURE 3 – Communication entre les composants de la plateforme

```
# BOSH
location /http-bind {
        proxy_pass      http://localhost:5280/http-bind;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header Host $http_host;
}
# xmpp websockets
location /xmpp-websocket {
        proxy_pass http://localhost:5280;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        tcp_nodelay on;
}
```

FIGURE 4 – Extrait de la configuration de `nginx`

Les communications entre le client et le serveur se font sur les ports suivants :
— 80 TCP - Connections HTTP qui servent à la vérification du certificat SSL
— 443 TCP - Accès à la plateforme Jitsi Meet (protocole HTTPS) à travers le serveur `nginx`
— 10000 UDP - Communications vidéo et audio vers `jitsi-videobridge`

3 Présentation de notre installation

3.1 Présentation de notre installation côté serveur

La plateforme Jitsi Meet est traditionnellement accessible sur un serveur à grande échelle via l'URL `https://meet.jit.si`. Pour réaliser notre étude, nous avons déployé nous-même notre environnement Jitsi Meet sur un serveur situé à l'IRISA, accessible par SSH afin de pouvoir par la suite collecter des métriques sur les ressources utilisées.

Nous avons effectué ce monitoring en utilisant la plateforme `netdata` : `https://github.com/netdata/netdata`. Cette plateforme est déployée sur le serveur où est hébergée la plateforme Jitsi, est accessible de l'extérieur grâce au serveur web `nginx`, et permet :
— De visualiser l'utilisation de l'ensemble des ressources du serveur au cours du temps
— De récupérer, grâce à son API, un document au format JSON contenant l'intégralité des données collectées pour un graphique (par exemple, l'utilisation du CPU) sur une période de temps.

`jitsi-videobridge` récupère des statistiques pendant les conférences, qui sont accessibles en exécutant une requête sur le port 8080. Il existe de plus une extension pour `netdata` qui permet de récupérer ces statistiques et de les intégrer aux données collectées par la plateforme `netdata` : `https://github.com/ctrlaltdel/netdata/blob/jitsi/collectors/python.d.plugin/jitsi/jitsi.chart.py`

En outre, nous avons mis en place une page JavaScript hébergée sur le serveur qui utilise la librairie ApexCharts, qui permet de visualiser les métriques collectées pour une ou plusieurs conférences données.

Pour optimiser le temps de chargement de cette page, nous avons écrit un script auxiliaire en Python. Ce script exécute une requête vers le serveur en suivant l'API `netdata`, afin de récupérer les dates des conférences ayant eu lieu. Pour détecter une conférence, le script parcourt les données `netdata` du graphique `jitsi_local.participants` (inclus dans l'extension pour Jitsi). Lorsque le nombre total de participants n'est pas nul, une conférence est comptabilisée. Ces dates sont enregistrées dans un fichier sur le serveur. Ensuite, le script parcourt les métriques collectées aux dates des conférences (CPU, bande passante, etc...) et enregistre un fichier au format JSON pour chaque conférence.

A l'aide d'une tâche `cron`, ce script est lancé toutes les heures. Ainsi, lorsque l'on charge la page JavaScript, la liste des conférences peut directement être obtenue, et lorsqu'on choisit les conférences dont on veut voir les métriques, il n'y a pas besoin d'utiliser l'API netdata, les points du graphique sont déjà précalculés pour chaque conférence.

Les graphes que nous avons suivi sont :
— L'utilisation du CPU liée à l'utilisateur (*user*) et de la RAM
— Le trafic réseau entrant et sortant
— Le nombre de *context switches* par seconde

— Le nombre d'instructions par seconde
— Le nombre d'accès cache et de *cache misses*

3.2 Présentation de notre installation côté client

3.2.1 Présentation générale

Pour pouvoir simuler des conférences avec un grand nombre de participants de manière automatisée, nous avons utilisé un programme dérivé du dépôt `https://github.com/elastest/elastest-webrtc-qoe-meter` qui fait partie de la plateforme de test Elastest. Ce programme Java compilé avec Maven utilise la librairie Selenium avec ChromeDriver pour piloter le navigateur Chromium / Google Chrome en mode *headless*, c'est à dire sans interface graphique. Ainsi, on peut simuler des connexions analogues à celles d'un utilisateur lambda se connectant à notre plateforme Jitsi Meet via Chromium (voir figure 5). En outre, en ajoutant une option lors de l'exécution de Chromium, il est possible de générer un flux vidéo factice.

Nous avons spécifiquement adapté le programme pour qu'il soit compatible avec Jitsi Meet. A l'aide de ce système, nous avons pu simuler des conférences avec un grand nombre de participants. Pour pouvoir réaliser ces tests avec une puissance CPU et une bande passante suffisantes, nous avons mis en service deux machines virtuelles, situées à l'IRISA, qui servent d'injecteurs de charge.

Nous avons en outre déployé `netdata` ainsi que notre page JavaScript sur une des machines virtuelles afin de quantifier l'utilisation des ressources du côté de la machine client lors d'une session de visio-conférence.

Finalement, le dernier problème qui se pose dans la réalisation de nos expériences est celui de la qualité de la vidéo / de l'audio reçus. En effet, si le nombre de participants connectés lors d'une session devient trop grand, la vidéo et l'audio peuvent diminuer en qualité ce qui peut se révéler très problématique. Le programme Java qui nous a servi à réaliser les tests provient d'une étude visant à évaluer le QoE, un indicateur subjectif de qualité vidéo [1]. En fait, lors d'une simulation de visio-conférence, notre programme injecte un code JavaScript sur la page de Jitsi Meet chargée par Chrome à l'aide d'une technique de Monkey Patching, qui permet de créer un objet RecordRTC (`https://github.com/muaz-khan/RecordRTC`) pour enregistrer un flux vidéo sortant ainsi qu'un flux entrant au format WebM lors du déroulement de la conférence. Il est possible, en comparant le flux sortant (référence) et le flux entrant, de déduire la perte de qualité du flux multimédia lors du trafic à l'aide d'algorithmes qui se basent sur certains indicateurs comme SSIM (Structural Similarity), PSNR (Peak Signal Noise Ration) ou encore PESQ (Perceptual Evaluation of Speech Quality) pour l'audio.

3.2.2 Calcul des indicateurs de qualité

Etant donné que dans un contexte classique d'utilisation d'un service de visio-conférence, une bonne qualité audio est toujours nécessaire, et la plupart du temps suffisante pour que le client ait une bonne opinion du service, dans le cadre de notre stage, nous nous sommes uniquement intéressés à la qualité de l'audio.

Pour quantifier la qualité d'un enregistrement audio, on s'intéresse à l'indicateur MOS (Mean Opinion Score). Le MOS est un score entre 1 et 5, qui, à l'origine, vise à évaluer la qualité de la restitution d'un enregistrement encodé par un codec donné. Ici, on cherche à

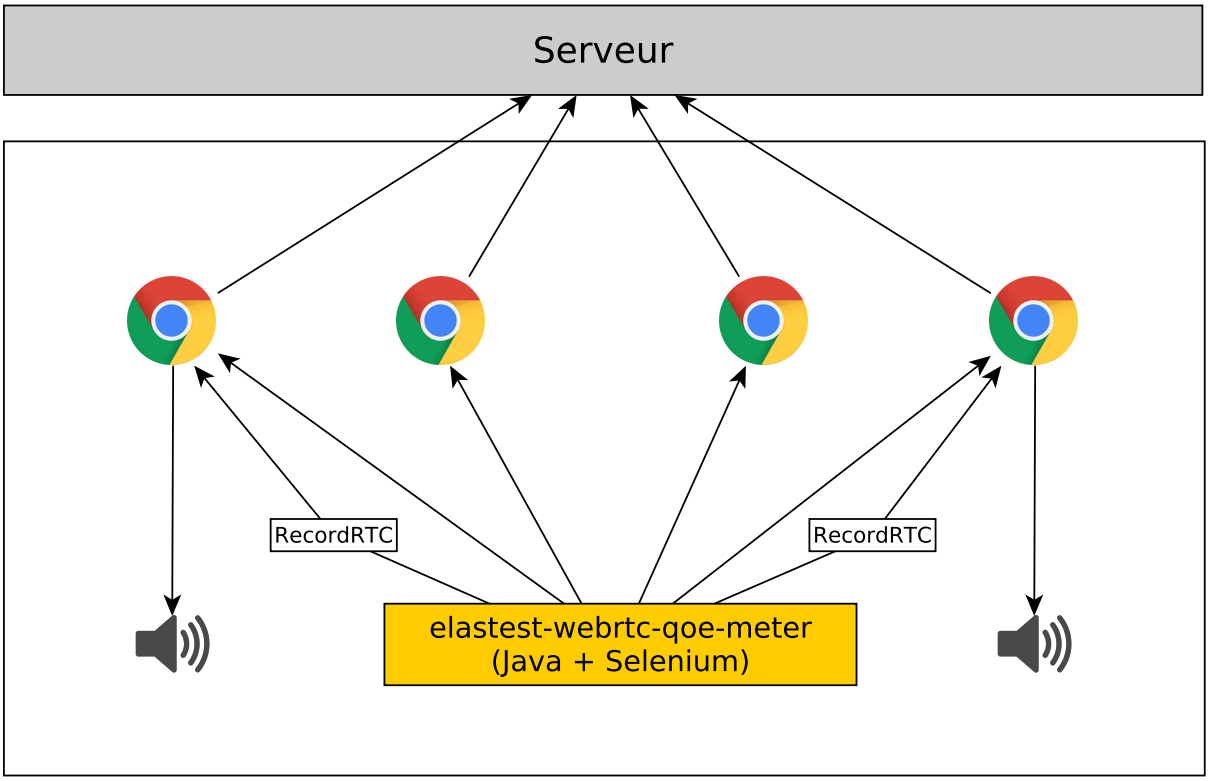


FIGURE 5 – L'installation côté client. L'objet RecordRTC injecté dans la page JavaScript chargée par Chrome sert à récupérer des enregistrements audio

estimer la qualité de l'enregistrement entrant par rapport à l'enregistrement sortant qui sert de référence. Le PESQ, par exemple, est un estimateur du MOS. Cependant, l'algorithme de calcul du PESQ se base sur la comparaison d'un enregistrement d'origine (de bonne qualité) avec l'enregistrement dont on veut mesurer la qualité (c'est un algorithme intrusif). Pour que les résultats soient fiables, il faut donc que les deux enregistrements soient parfaitement alignés frame à frame lors de la comparaison, ce qui s'est révélé en pratique extrêmement difficile à réaliser, d'autant plus que des décalages peuvent survenir lors de l'enregistrement.

C'est pourquoi nous nous sommes intéressés à des algorithmes non-intrusifs, c'est à dire des algorithmes qui sont capables d'estimer le MOS sans avoir de flux de référence. On comparera alors le MOS de la vidéo sortante avec celui de la vidéo entrante, de manière à estimer la perte de qualité audio. Les principaux algorithmes non-intrusifs utilisent des réseaux de neurones. Le principe de ces algorithmes est de s'entraîner sur un grand nombre d'échantillons, dont on a préalablement déterminé le MOS à partir d'un enregistrement de référence (par exemple avec PESQ), pour pouvoir ensuite estimer le MOS d'un enregistrement inconnu. L'implémentation que nous avons utilisé est MOSNet [3] (`https://github.com/lochenchou/MOSNet`).

3.3 Scénarios d'expérimentation

Nous avons effectué plusieurs types de tests afin d'essayer notre installation dans des conditions différentes. Le flux vidéo est généré à partir d'un échantillon vidéo "réaliste", c'est à dire une discussion telle qu'on peut avoir dans le cadre d'une visioconférence. Plus

exactement, nous avons trois types de tests (voir figure 6) :

— Dans le premier, tous les participants sont dans la même session
— Dans le second, les participants sont répartis dans différentes sessions de 2 participants (gérées en mode P2P).
— Dans le troisième, les participants sont répartis dans différentes sessions de 4 participants (gérées en mode SFU)

Dans le premier cas, on s'attend à avoir une consommation de ressources (CPU, bande passante) plus importante car tous les flux vidéo doivent transiter entre tous les participants (contrairement à une répartition en plusieurs sessions car deux participants de deux sessions différentes n'échangeront pas leurs flux).

On s'attend à avoir une consommation encore moindre dans le deuxième cas, car les flux vidéo n'auront pas à transiter par le serveur en mode P2P.

Pour réaliser nos tests, nous avons programmé des tâches `cron` sur les deux VM. Environ toutes les 8 minutes, un test parmi les trois types est lancé sur les deux machines en même temps (en alternant les trois), avec un nombre total de participants compris entre 4 et 20. Détaillons le fonctionnement de chaque type de test.

— Pour le premier type de test qui consiste à placer tous les participants dans une seule grosse session, on calcule un nombre n entre 2 et 10 qui correspond aux heures de la date du test modulo 9, plus 2 (par exemple, si le test est effectué à 14 :15, le nombre tiré sera 7). Ce nombre sera le même sur les deux machines, et chaque machine génèrera n participants qui se rejoindront dans la même session pour réaliser un test à $2n$ participants.
— Pour le second type de test, on prend cette fois un autre nombre n qui correspond aux heures modulo 4 plus 2, donc un nombre entre 2 et 5, et chaque machine génèrera n conférences de deux personnes. En tout, on obtient alors $2n$ conférences de 2 participants, donc un test à $4n$ participants (au total, 8 à 20 participants).
— Pour le troisième type de test, on procède de la même manière, excepté que les n conférences générées par chaque machine se rejoignent pour former finalement n conférences à 4 participants.

A la suite de chaque test, on enregistre les audios capturés pour pouvoir ensuite estimer leur qualité à l'aide de MOSNet. Ce procédé permet d'obtenir des expériences assez variées et nombreuses pour pouvoir par la suite dégager une tendance d'utilisation des ressources et de la qualité de la vidéo pour chaque scénario, en fonction du nombre de participants.

4 Résultats obtenus

Tout d'abord, on peut remarquer que l'application Jitsi Meet est plutôt gourmande en ressources du côté du client. En effet, on a une utilisation CPU d'environ 45% pour quatre clients connectés à l'application sur une des machines virtuelles, et environ 70% pour six clients. Les machines virtuelles utilisées dans notre étude ont un processeur Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz, 4 coeurs. C'est ce qui nous a limité en pratique pour pouvoir faire des simulations avec un nombre total de participants supérieur à 20.

De plus, nous avons remarqué que du côté du serveur, la plateforme Jitsi ne consomme pratiquement pas plus de mémoire lors des conférences : il n'y a presque pas de différence entre le pourcentage de RAM utilisée avec 4 participants et le pourcentage de RAM utilisée

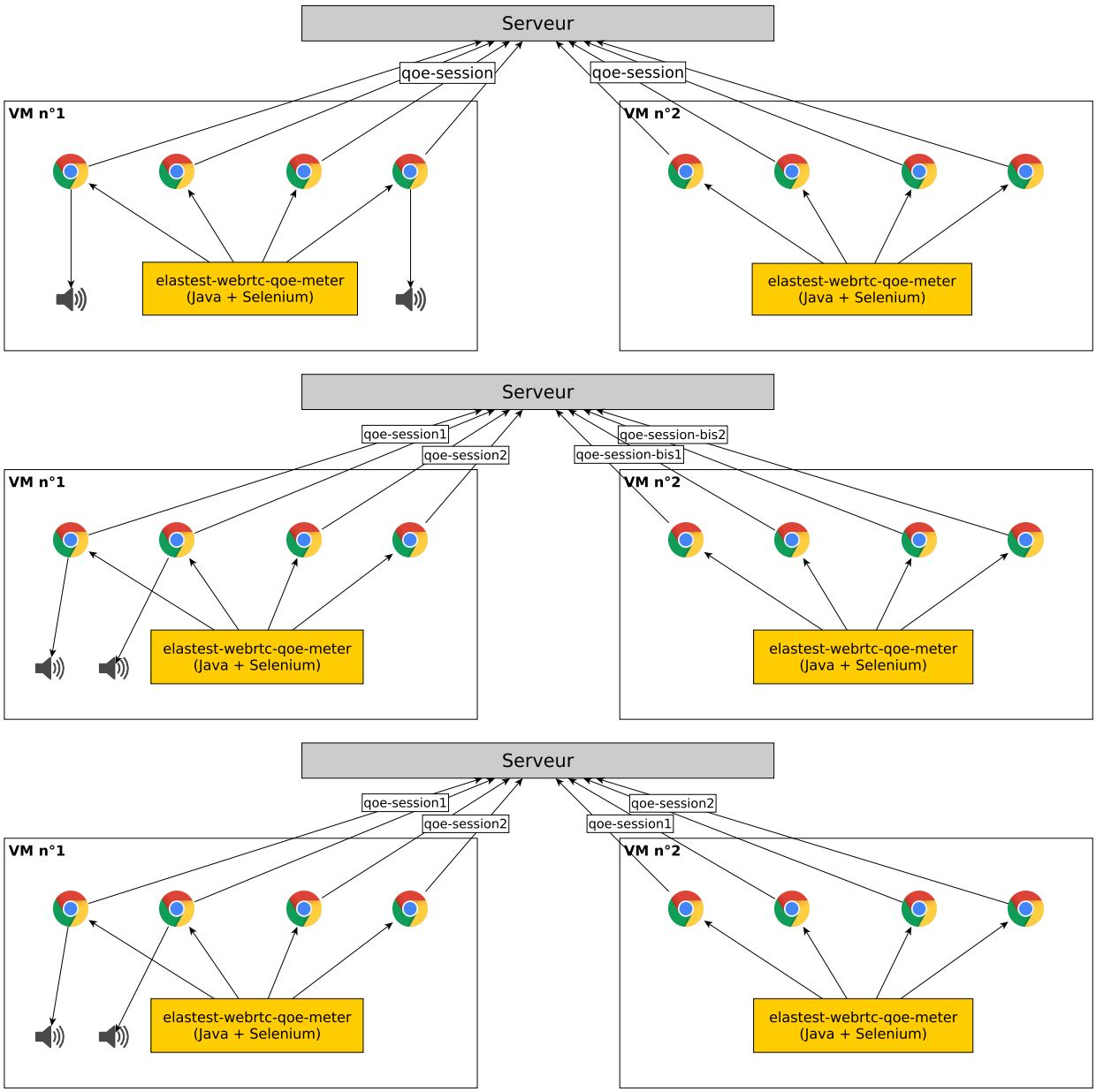
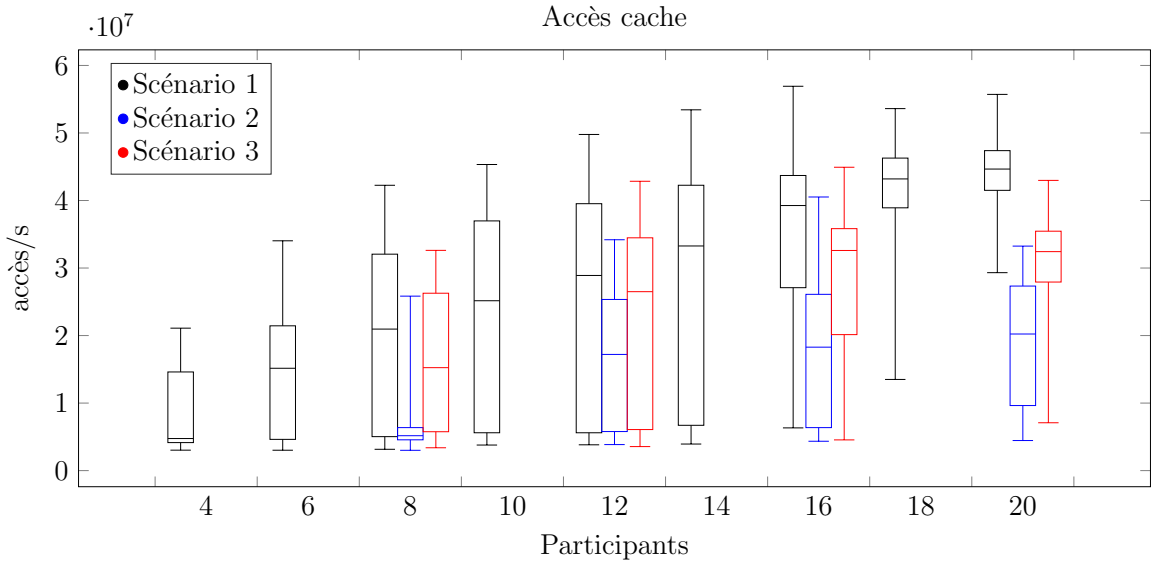
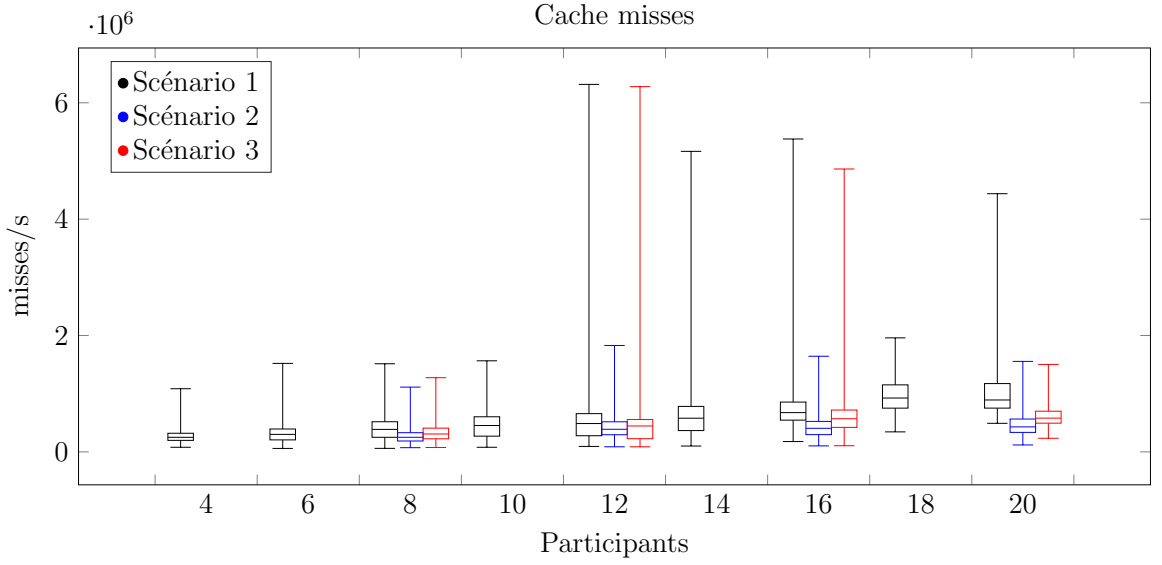


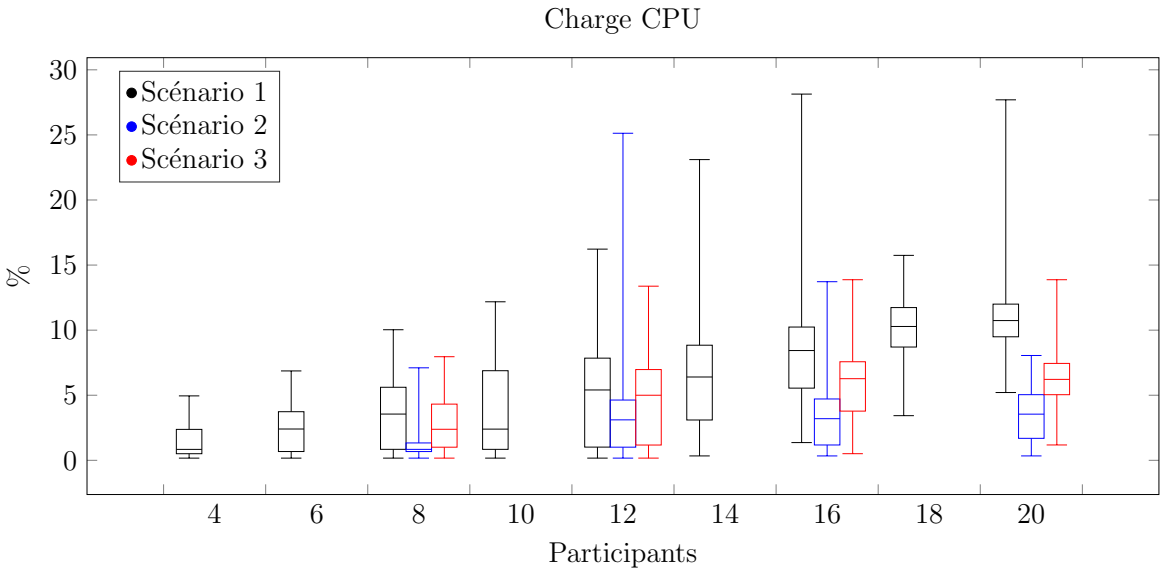
FIGURE 6 – Les trois types de tests possibles. En haut, tous les participants participent à la même session (`qoe-session`). Au milieu, ils sont séparés dans des sessions de 2 participants. En bas, ils sont séparés dans des sessions de 4 participants.

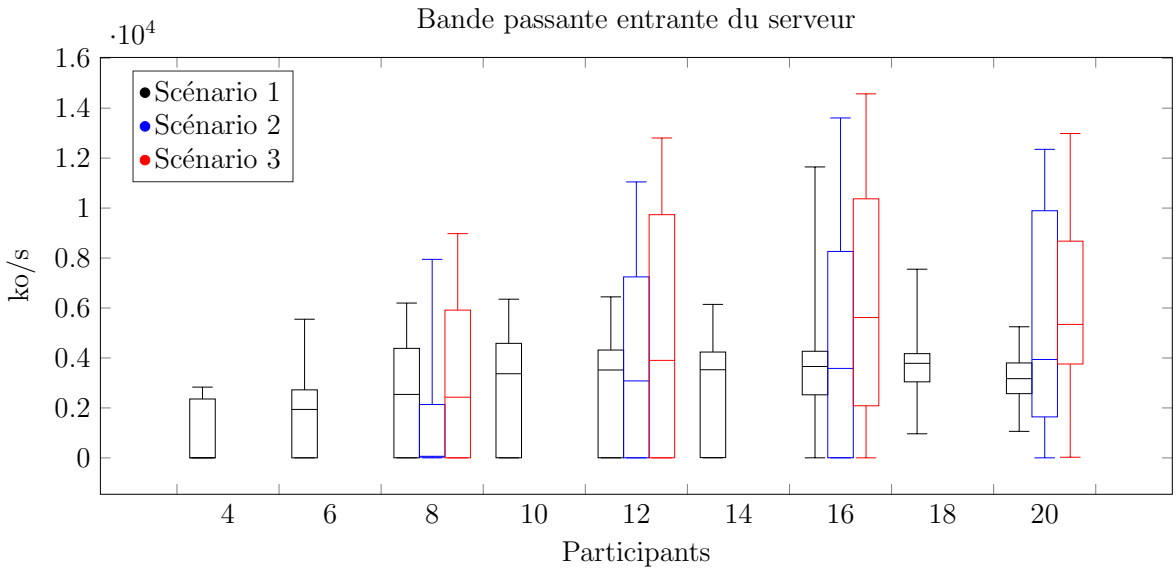
avec 20 participants. Par conséquent, nous n'avons pas collecté de données sur l'utilisation de la mémoire vive.

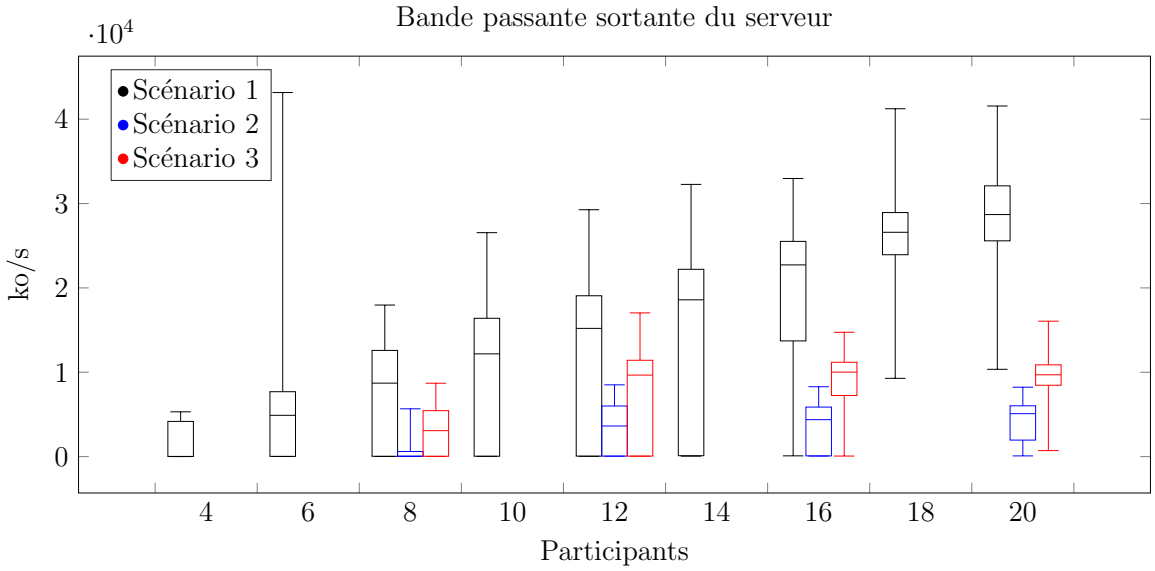
Les données collectées pour les trois scénarios sont représentées ci-dessous sous forme de diagrammes en boîte.

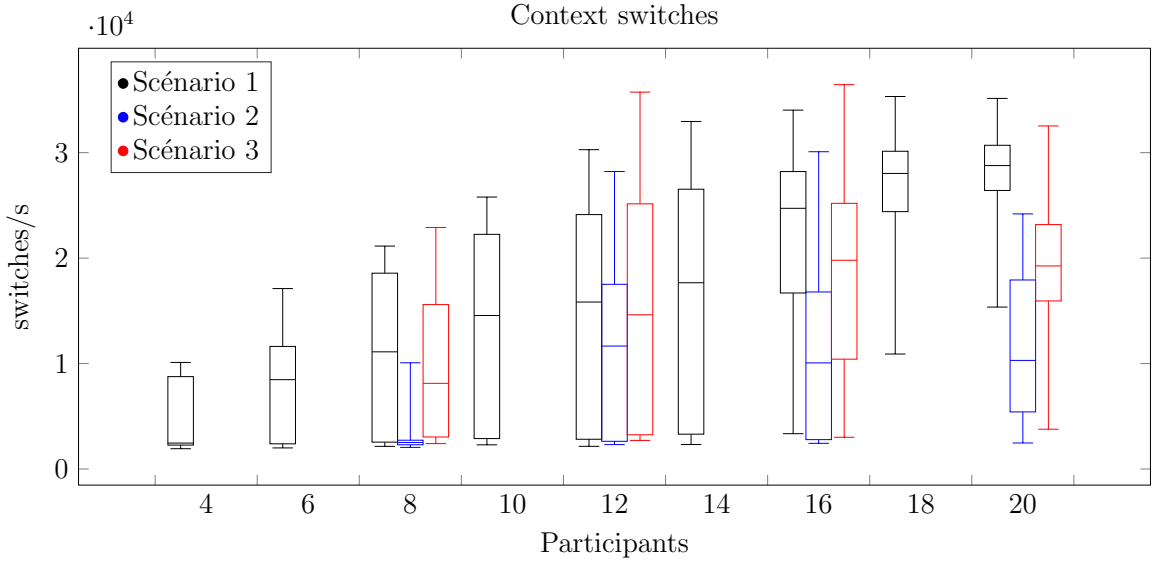

Accès cache
$\cdot 10^7$
accès/s
Scénario 1
Scénario 2
Scénario 3
0
1
2
3
4
5
6
4
6
8
10
12
14
16
18
20
Participants

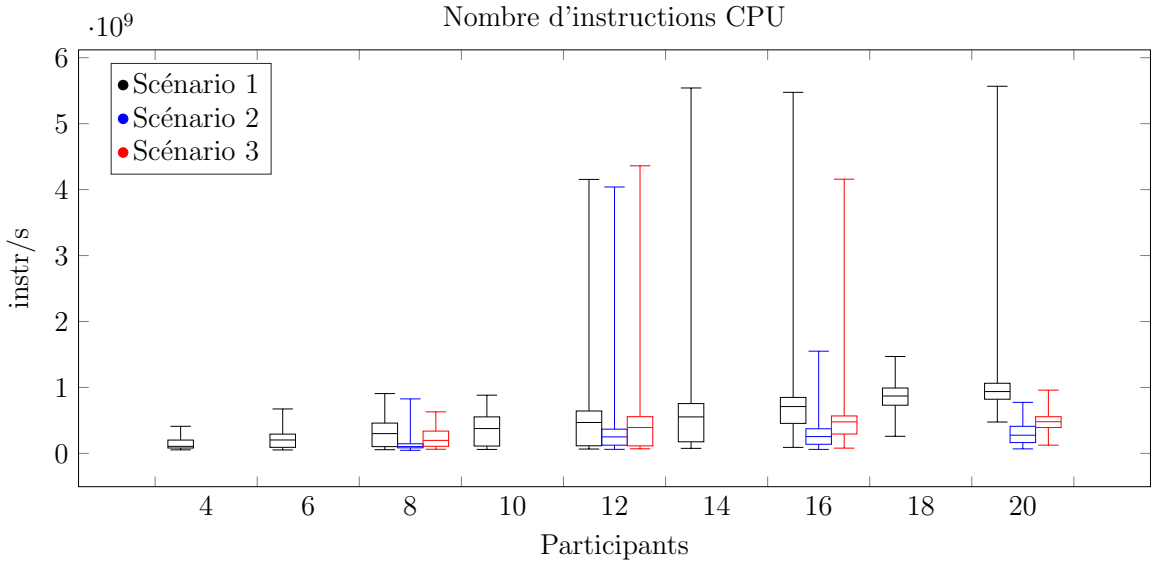

Cache misses
$\cdot 10^6$
misses/s
Scénario 1
Scénario 2
Scénario 3
0
2
4
6
4
6
8
10
12
14
16
18
20
Participants


Charge CPU
%
Scénario 1
Scénario 2
Scénario 3
0
5
10
15
20
25
30
4
6
8
10
12
14
16
18
20
Participants


Bande passante entrante du serveur
$\cdot 10^4$
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
ko/s
Scénario 1
Scénario 2
Scénario 3
4
6
8
10
12
14
16
18
20
Participants


Bande passante sortante du serveur
$\cdot 10^4$
4
3
2
1
0
ko/s
Scénario 1
Scénario 2
Scénario 3
4
6
8
10
12
14
16
18
20
Participants


Context switches
$\cdot 10^4$
3
2
1
0
switches/s
Scénario 1
Scénario 2
Scénario 3
4
6
8
10
12
14
16
18
20
Participants



Globalement, pour tous les indicateurs, excepté la bande passante entrante. on observe la même tendance : le scénario 1 (tous les participants dans la même session) est plus gourmand en ressources que le scénario 3 (conférences de 4 personnes), lui-même plus gourmand que le scénario 2 (conférences de 2 personnes).

Le taux de cache misses est, tous scénarios et tous nombres de participants confondus, toujours situé autour de 1-2% ce qui est un bon ratio. Il y a peu d'améliorations à proposer sur ce point. La fréquence des context switches est quant à elle particulièrement élevée lors de l'utilisation de Jitsi Meet, pouvant atteindre les 30000 par seconde avec une vingtaine de participants dans la même session. Ces context switches ont probablement lieu lors de l'envoi de signaux par `jicofo`, en particulier lors des changements de l'utilisateur ayant la parole.

Le mode P2P offre un gain très intéressant en termes de scalabilité : comme les flux multimédia ne transitent pas par le serveur, la seule charge pour le serveur est de distribuer l'accès à l'application Jitsi Meet à tous les participants, ce qui est très peu coûteux comparé à l'encodage/décodage des flux vidéo. Le problème de ce mode est qu'il reste plus consommateur de bande passante côté client pour des conférences de trois participants ou plus.

Concernant la bande passante entrante, il est prévisible que la différence entre les scénarios soit moindre dans la mesure où avec n participants, il y a toujours n flux vidéo qui arrivent au serveur peu importe la configuration. En revanche, au niveau de la bande passante sortante, la différence est encore plus marquée : avec le mode SFU, le nombre de flux vidéo pour n participants dans la même session est en $\Theta(n^2)$, alors qu'avec n sessions de 4 participants, le nombre de flux croît en $\Theta(n)$.

On peut en outre affirmer qu'en pratique, c'est la bande passante qui limite la scalabilité de la plateforme Jitsi Meet : même avec une vingtaine de participants, l'utilisation CPU est de l'ordre de 10%, et le nombre d'instructions par seconde autour de 1.10^9 donc clairement en deçà de la capacité d'un processeur comme le nôtre (6 coeurs à 1,6 GHz). Le serveur est donc loin d'être saturé du point de vue de sa capacité de calcul. Cependant, avec le même nombre de participants, la bande passante sortante du serveur est d'environ 30 Mo/s, soit 240 mégabits par seconde, soit près d'un quart d'une bande passante de bonne qualité (1 gigabit par seconde).

Concernant la qualité des enregistrements audio, on remarque qu'il n'y a pratiquement pas

de dégradation jusqu'à 20 participants. Pour les trois scénarios confondus, la perte de MOS maximale entre l'audio sortant et l'audio entrant que l'on ait relevé est d'environ 0.4. Parfois, MOSNet donne même un meilleur score à l'enregistrement entrant qu'à l'enregistrement sortant. Ces résultats sont toutefois à relativiser dans la mesure où une perte de qualité peut se produire dès l'enregistrement par RecordRTC, et peut fausser les résultats.

5 Conclusion et améliorations possibles

Tous ces résultats nous amènent à conclure que pour améliorer la scalabilité de Jitsi Meet, il faudrait se focaliser sur la limitation de la bande passante nécessaire à l'utilisation. Le développement d'une stratégie MCU serait un bon choix : bien qu'une telle stratégie soit plus gourmande en charge CPU (ce qui n'est pas un problème majeur étant donné que la charge CPU de Jitsi est modérée), elle consomme significativement moins de bande passante.

Pour avoir des résultats plus intéressants sur la qualité de l'audio reçue, il serait judicieux de faire des essais à plus grande échelle encore pour savoir jusqu'à quel niveau de charge un serveur Jitsi peut assurer une bonne qualité aux utilisateurs. Le principal obstacle pour réaliser ces essais est la bande passante disponible, qui limite rapidement le flux de données qui peut arriver au serveur sans pour autant le surcharger en termes de ressources. Des tests avec des conférences de 4 personnes (scénario 3) sont ici les plus adaptés, car ils testent la gestion des flux vidéo en mode SFU (avec `jitsi-videobridge`) sans pour autant consommer trop de bande passante.

L'équipe d'Andrea Lottarini a mené une étude approfondie des codecs vidéo [4], qui vise à comparer l'efficacité de différents codecs en termes de rapidité, de qualité et de consommation de bande passante en créant un benchmark qui vise à englober la plupart des types de vidéos que l'on peut trouver sur des plateformes comme YouTube ou Netflix. Le développement d'une étude similaire, en créant un autre benchmark plus fidèle aux flux vidéo échangés lors d'une session de visio-conférence, et en se focalisant sur l'optimisation en termes de consommation de bande passante, est quelque chose à considérer à la suite de ce stage dans l'optique de l'amélioration de la scalabilité des systèmes de visio-conférence.

Sur le long terme, il serait tentant de faire évoluer les protocoles traditionnels de transmission des flux multimédia. Des résultats prometteurs ont été mis en avant dans la thèse de Jacques Samain [6] grâce aux réseaux centrés sur l'information (Information-Centric Networking, ICN) qui viendraient remplacer les protocoles basés sur TCP/IP. Le paradigme d'ICN est de se focaliser sur les données transmises entre les machines, et non sur la connexion entre les machines. Avec ICN, plutôt que d'établir une connexion avec un serveur, puis transmettre un flux de données, on envoie les données vers le serveur, qui sera par la suite notifié de cet envoi, et ce dernier récupèrera les données. Cette approche a deux avantages principaux. Tout d'abord, il n'y a pas besoin de maintenir des connexions avec des machines qui ne sont pas actives, ce qui permet d'économiser de la bande passante. En outre, elle facilite la transmission des données en cas de mauvaise connexion voire de coupures de connexion, étant donné qu'il n'y a pas de connexion entre les hôtes. Le protocole WebRTC serait alors remplacé par une solution plus efficace qui exploite les particularités d'ICN de manière à ce que seuls les participants actifs (par exemple, dans une session de visio-conférence), c'est à dire ceux qui envoient des données, n'utilisent la bande passante du serveur. Cela réduirait significativement l'importance du principal frein à la scalabilité des systèmes de visio-conférence, à savoir

la limite de bande passante.

En conclusion, ce stage nous permet d'affirmer que c'est l'utilisation de la bande passante qui est la contrainte la plus forte pour rendre un système de visio-conférence le plus scalable possible. Ces résultats nous ouvrent ainsi des portes vers l'étude de sujets beaucoup plus vastes sur la théorie de la transmission de l'information. L'aboutissement de ce stage s'inscrit donc dans le cadre d'un domaine très actif de la recherche : en effet, dans la mesure où le volume d'informations et notamment de flux vidéo transmis sur Internet augmente de manière vertigineuse avec le temps, les méthodes pour échanger des informations sont amenées à être en perpétuelle évolution.

Lors du stage, j'ai rencontré plusieurs problèmes. Tout d'abord, j'ai eu des difficultés pour bien configurer les plateformes Jitsi et `netdata`, et plus généralement pour résoudre les problèmes techniques concernant le serveur. Ensuite, au début du stage, j'ai été bloqué par l'indisponibilité des serveurs de l'IRISA puis par celle des machines virtuelles. Enfin, un dernier problème d'attente s'est posé lors de la collecte des résultats : en effet, pour avoir suffisamment de simulations, il faut au moins 24h de tests, et j'ai dû réaliser ces séries de tests à plusieurs reprises pour affiner les résultats.

Références

[1] Boni García, Francisco Gortázar, Micael Gallego, and Andrew Hines. Assessment of qoe for video and audio in webrtc applications using full-reference models. *Electronics*, 9 :462, 03 2020.

[2] Thierry Kauffmann. Architecture des applications modernes de visioconférence. `https://www.arawa.fr/2020/04/30/architecture-des-applications-modernes-de-visioconference`, 30 avril 2020.

[3] Chen-Chou Lo, Szu-Wei Fu, Wen-Chin Huang, Xin Wang, Junichi Yamagishi, Yu Tsao, and Hsin-Min Wang. Mosnet : Deep learning based objective assessment for voice conversion. In *Proc. Interspeech 2019*, 2019.

[4] Andrea Lottarini, Alex Ramirez, Joel Coburn, Martha Kim, Parthasarathy Ranganathan, Daniel Stodolsky, and Mark Wachsler. vbench : Benchmarking video transcoding in the cloud. *ACM SIGPLAN Notices*, 53 :797–809, 03 2018.

[5] Ian Paterson, Dave Smith, Peter Saint-Andre, Jack Moffitt, Lance Stout, and Winfried Tilanus. Xep-0124 : Bidirectional-streams over synchronous http (bosh). `https://xmpp.org/extensions/xep-0124.html#appendix-docinfo`.

[6] Jacques Samain. *Improving quality of experience in multimedia streaming by leveraging Information-Centric Networking*. PhD thesis, Télécom ParisTech, 03 2019.

A Contexte institutionnel et social du stage

J'ai été accueilli par l'équipe Whisper de l'INRIA pour réaliser ce stage, et encadré par Gilles Muller, membre de l'INRIA, qui est à la tête de l'équipe. Cette équipe est spécialisée dans l'étude des systèmes d'exploitation.

Ce stage s'est déroulé, conformément aux mesures liées au contexte sanitaire, en visio-conférence. Nous n'avons malheureusement pas pu nous voir en présentiel au laboratoire. Nous nous réunissions, avec l'équipe, deux fois par semaine, pour faire le point sur ce qui a été fait, les pistes explorées et les résultats obtenus. Mes encadrants me donnaient des indications pour poursuivre la recherche jusqu'à la réunion suivante. Nous pouvions échanger sur un serveur Slack pour partager les résultats, les papiers de recherche intéressants dans le cadre du stage ainsi que pour les problèmes divers liés au déploiement de la plateforme Jitsi sur les machines. J'ai rassemblé tous les codes sources utilisés sur le dépôt Git public suivant : `https://gitlab.aliens-lyon.fr/rlaspina/stagel3`.