SRTP: Simple Reliable Transport Protocol

Compilé : 20 mars 2026

1 Description du projet

Une célèbre marque de chaussures de course, *C'Hokayy*, veut prochainement se lancer sur le marché des montres de course afin de diversifier son offre. La particularité de cette montre ? Un modèle de *Large Language Model* (LLM) directement embarqué sur l'appareil ! Et qui plus est, ce modèle est régulièrement mis à jour pendant la course afin de garantir sa précision au fil du temps. La communication entre l'appareil et le serveur distant pour télécharger les nouveaux modèles requiert donc un protocole de transport fiable.

Puisque ces modèles de LLM sont très lourds et que la capacité de stockage de ces montres est limitée, l'entreprise souhaite implémenter un protocole de transport fiable et simplifié : le *Simple Reliable Transport Protocol (SRTP)*, basé sur des segments UDP. L'entreprise *C'Hokayy* désire une implémentation en Python pour garder le code simple et fonctionnel — de toute façon, son LLM est capable de le réécrire, si besoin, en n'importe quel langage, en quelques minutes.

SRTP permettra de réaliser des transferts fiables de fichiers et utilisera la stratégie du **selective repeat** [1]. Comme l'entraînement du modèle LLM coûte déjà assez cher à développer, l'entreprise ne souhaite pas équiper ses montres d'un support IPv4, vu le coût pour obtenir une telle adresse sur chaque appareil. SRTP devra donc fonctionner au-dessus d'IPv6, où il est beaucoup plus facile d'obtenir des adresses à moindre coût.

L'entreprise veut utiliser la version 0.9 de HTTP au-dessus de SRTP. Ce projet doit être réalisé par **groupes de trois**. Deux programmes sont à réaliser, permettant de faire un transfert de données entre deux machines distantes.

2 Spécifications

2.1 Protocole

Le format des segments du protocole est visible sur la figure 1. Ils se composent des champs dans l'ordre suivant :

Type Ce champ est encodé sur 2 bits. Il indique le type du paquet, trois types sont possibles :

(i) `PTYPE_DATA` = 1, indique un paquet contenant des données ;

(ii) `PTYPE_ACK` = 2, indique un paquet d'acquittement de données reçues ;

(iii) `PTYPE_SACK` = 3, indique un paquet d'acquittement de données sélectifs (c.f section 2.3.1).

Un paquet avec un autre type **DOIT** être ignoré ;

1. Le selective repeat implique uniquement que le `python3 client.py` accepte les paquets hors séquence et les stocke dans un buffer s'ils sont dans la fenêtre de réception. Par contre, ce protocole-ci ne permet pas obligatoirement au `python3 client.py` d'indiquer au `python3 server.py` quels sont les paquets hors-séquence qu'il a reçus.

Window Ce champ est encodé sur 6 bits, et varie donc dans l'intervalle [0, 63]. Il indique la taille de la fenêtre de réception de l'émetteur de ce paquet. Cette valeur indique le nombre de places vides dans le buffer de réception de l'émetteur du segment, et peut varier au cours du temps. Si un émetteur n'a pas de buffer de réception, cette valeur **DOIT** être mise à 0. Si le champ **Window** du dernier paquet provenant du destinataire vallait 0, alors un émetteur ne peut envoyer de nouvelles données tant qu'un acquit pour le paquet précédent n'a pas été reçu. Lors de la création d'un nouvelle connexion, l'émetteur initiant la connexion **DOIT** considérer que le destinataire avait annoncé une valeur initiale de **Window** valant 1.

Length Ce champ est encodé sur 13 bits en network-byte order, et sa valeur varie dans l'intervalle [0, 1024]. Il indique le nombre d'octets de données du payload. Un paquet `PTYPE_DATA` avec ce champ à 0 et dont le numéro de séquence correspond au dernier numéro d'acquittement envoyé par le destinataire signifie que le transfert est terminé. Si ce champ vaut plus que 1024, le paquet **DOIT** être ignoré ;

Seqnum Ce champ est encodé sur 11 bits, et sa valeur varie dans l'intervalle [0, 2047]. Sa signification dépend du type du paquet.

`PTYPE_DATA` Il correspond au numéro de séquence de ce paquet de données. Le premier segment d'une connexion a le numéro de séquence 0. Si le numéro de séquence ne rentre pas dans la fenêtre des numéros de séquence autorisés par le destinataire, celui-ci **DOIT** ignorer le paquet ;

`PTYPE_ACK` Il correspond au numéro de séquence du prochain numéro de séquence attendu (c-à-d (le dernier numéro de séquence + 1) % 2^{11}). Il est donc possible d'envoyer un seul paquet `PTYPE_ACK` qui sert d'acquittement pour plusieurs paquets `PTYPE_DATA` (principe des acquis cumulatifs) ;

Lorsque l'émetteur atteint le numéro de séquence 2047, il recommence à 0 ;

Timestamp Ce champs est encodé sur 4 octets, et représente une valeur opaque donc sans endianness particulière. Pour chaque paquet `PTYPE_DATA`, l'émetteur d'un paquet choisit une valeur à mettre dans ce champ. Lorsque le destinataire envoie un paquet `PTYPE_ACK` il indique dans ce champ la valeur du champ **Timestamp** du **dernier** paquet `PTYPE_DATA` reçu. La signification de la valeur est laissée libre aux implémenteurs.

CRC1 Ce champ est encodé sur 4 octets, en network byte-order. Ce champ contient le résultat de l'application de la fonction CRC32 [2] au header, juste avant qu'il ne soit envoyé sur le réseau. À la réception d'un paquet, cette fonction doit être recalculée sur le header, et le paquet **DOIT** être ignoré si les deux valeurs diffèrent.

Payload Ce champ contient au maximum 1024 octets. Il contient les données transportées par le protocole. Sa taille est définie par le champ **Length**. Le **payload** de chaque paquet est considéré comme un bloc unique, occupant une place dans le buffer de réception. En d'autres termes, la réception d'un paquet de type `PTYPE_DATA` avec un **Payload** non vide réduit la fenêtre de réception de 1 tant que la donnée n'est pas consommée par l'application.

CRC2 Ce champ est encodé sur 4 octets, en network byte-order. Ce champ contient le résultat de l'application de la fonction CRC32 de l'éventuel **Payload**, juste avant qu'il ne soit envoyé sur le réseau. Ce champ n'est présent que si le paquet contient un **Payload** et n'a pas été tronqué. À la réception d'un paquet contenant ce champ, cette fonction doit être recalculée sur le **Payload**, et le paquet **DOIT** être ignoré si les deux valeurs diffèrent.

2. L'implémentation la plus courante de cette fonction se trouve sur https://docs.python.org/3/library/zlib.html, mais de nombreuses autres existent.

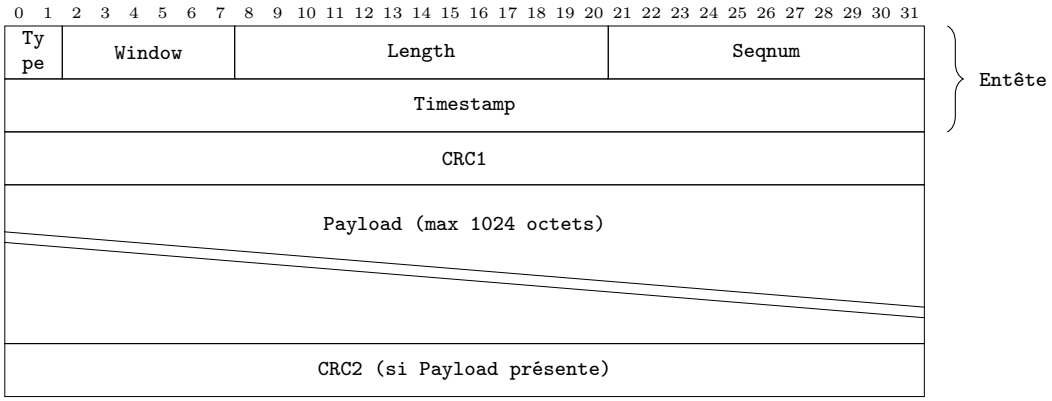

FIGURE 1 – Format des segments du protocole

Puisque *C'Hokayy* vise les aventurier.e.s les plus intrépides comme les coureur.e.s du dimanche, le protocole SRTP devra fonctionner correctement dans le modèle réseau suivant :

1. Un segment de données envoyé par un hôte est reçu **au plus une fois** (pertes mais pas de duplication) ;
2. Le réseau peut **corrompre** les segments de données de façon aléatoire ;
3. Le réseau peut **tronquer** le payload des paquets de façon aléatoire ;
4. Soient deux paquets, P_1 et P_2, si P_1 est envoyé avant P_2, il n'y a **pas de garantie** concernant l'**ordre** dans lequel ces paquets seront reçus à la destination ;
5. La **latence** du réseau pour acheminer un paquet varie dans l'intervalle **[0,2000]** (ms).

2.2 HTTP comme protocole applicatif

Plusieurs modèles LLM sont disponibles sur les serveurs de *C'Hokayy* en fonction du modèle de la montre et des performances du modèle choisi — un modèle plus puissant consomme plus de batterie ; l'entreprise propose donc aussi des modèles économes en énergie. L'entreprise propose donc d'utiliser HTTP 0.9 pour sélectionner le modèle à télécharger sur la montre. HTTP 0.9 est une version simplifiée du protocole HTTP. Elle est aujourd'hui considérée comme obsolète pour les communications sur Internet, mais reste largement suffisante dans le contexte de *C'Hokayy*.

Dans HTTP 0.9, le client envoie sa requête sous la forme `GET /chemin/vers/le/fichier` encodée en ASCII. Le serveur répond directement avec le contenu du fichier demandé (potentiellement au format binaire). Dans le cas où le fichier demandé n'existe pas, le serveur répond par un paquet *vide* (**Length** = 0), indiquant la fin de la connexion. Dans HTTP 0.9, les headers ne sont pas supportés, veuillez ne pas en mettre.

2.3 Extension

Vous pouvez étendre SRTP par deux extensions : les acquittements sélectifs et la gestion de plusieurs clients simultanés. Ces deux extensions seront évidemment prises en compte lors de l'évaluation de votre protocole.

2.3.1 Acquittements sélectifs

Afin de minimiser la bande passante consommée en cas de pertes, l'entreprise propose d'étendre légèrement SRTP pour y ajouter le support des acquittements sélectifs. En plus de contenir le prochain numéro de séquence attendu *en séquence* (c'est-à-dire sans trou), un

acquittement sélectif ajoute aussi les numéros de séquence reçus hors-séquence, permettant à l'envoyeur d'éviter de renvoyer des paquets déjà reçus.

Un acquittement sélectif est de type `PTYPE_SACK`. La signification de ce type de paquet est similaire à celle des paquets de type `PTYPE_ACK`, à l'exception de :

Length Ce champ indique toujours la taille du **Payload**. Mais contrairement aux paquets de type `PTYPE_ACK`, la valeur de ce champ n'est pas nulle, ce qui indique qu'il y a des acquittements sélectifs. Puisqu'un paquet de type `PTYPE_SACK` avec une longueur nulle est équivalent à un paquet de type `PTYPE_ACK`, l'émetteur de ces acquittements **DOIT** utiliser un paquet de type `PTYPE_ACK` s'il n'y a pas d'acquittement sélectif à envoyer.

Payload Ce champ contient les acquittements sélectifs, en plus du prochain numéro de séquence attendu déjà contenu dans le champ **Seqnum** (similaire aux paquets de type `PTYPE_ACK`). Contrairement aux acquittements sélectifs de TCP qui retournent des étendues de **Seqnum** [1], cette extension ajoutera, en tant que **Payload**, les **Seqnum** reçus hors séquence. Comme le **Payload** est limité à 1024 octets et que chaque **Seqnum** est encodé sur 11 bits, nous avons un total de $\lfloor \frac{1024*8}{11} \rfloor = 744$ **Seqnum** reçus hors séquence dans chaque paquet de type `PTYPE_SACK`. Au cas où le nombre de **Seqnum** encodés dans le Payload ne permet pas d'obtenir un multiple de 4 octets, l'extension **DOIT** ajouter des bits de padding à la fin du **Payload**.

Si vous n'implémentez pas l'extension : vous devez traiter les paquets `PTYPE_SACK` comme des paquets de type `PTYPE_ACK`, c'est-à-dire récupérer la valeur du champ Seqnum comme le prochain numéro de séquence attendu.

2.3.2 Gestion de plusieurs clients en simultané

En pratique, un seul serveur peut devoir transmettre des fichiers à plusieurs clients en même temps (soit le même fichier, soit des fichiers différents, selon les modèles de montres des clients). Cette deuxième extension vise à permettre au serveur de gérer simultanément plusieurs connexions avec des clients. Plus précisément, cela signifie que si (par exemple) trois clients exécutent chacun une requête quasi simultanément, le troisième client ne devra pas attendre la complétion totale des requêtes des deux premiers clients pour recevoir le fichier.

2.4 Programmes

L'implémentation du protocole devra permettre d'effectuer un transfert de données unidirectionnel au moyen de deux programmes, `python3 server.py` et `python3 client.py`. Cette implémentation **DOIT** fonctionner sur les ordinateurs **Linux** de la salle Intel, bâtiment Réaumur [3].

Le programme du serveur se lance avec la commande suivante :
`>$ python3 server.py hostname port`, avec `hostname` étant le nom du domaine ou l'adresse IPv6 sur laquelle le serveur écoute les requêtes entrantes des clients, et `port` le numéro de port UDP sur lequel le serveur s'attache.

Le client doit se lancer avec la commande :
`>$ python3 client.py servername`, où `servername` a la forme `http://hostname:port/path/to/file`, où `hostname` et `port` ont la même signification que pour le serveur, et `path/to/file` est le chemin relatif du fichier à récupérer.

Le serveur doit supporter l'argument optionnel suivant :
`--root directory`, où `directory` est le dossier racine depuis lequel le serveur récupère le fichier demandé par le client. Par défaut, le dossier racine utilisé correspond à `.`, c'est-à-dire le

3. Les machines peuvent être accédées via SSH ; consultez https://wiki.student.info.ucl.ac.be/Mat%c3%a9riel/SalleIntel

dossier depuis lequel `python3 server.py` est lancé. En d'autres termes, le fichier demandé par le client devra correspondre pour le serveur au fichier se trouvant à la localisation retournée par `os.path.join(directory, path/to/file)`.

Enfin, le client doit supporter l'argument optionnel suivant :
`--save location` où `location` correspond au chemin et au nom du fichier où stocker les données reçues. Par défaut, le fichier est enregistré dans le dossier courant sous le nom `llm.model`.

Les deux programmes doivent utiliser la **sortie d'erreur standard** s'ils veulent afficher des informations à l'utilisateur.

Voici quelques exemples pour lancer les programmes :

`python3 server.py ::1 8080`	Le serveur écoute sur la même machine que le client (`::1`) sur le port 8080, et retourne les fichiers demandés par le client depuis le dossier courant (`.`).
`python3 server.py --root /home/www ::1 8080`	Idem, mais en cherchant les fichiers depuis `/home/www`. Par exemple, si le client effectue la requête `GET /llm/small`, cela correspond au fichier se trouvant sur la machine du serveur dans `/home/www/llm/small`.
`python3 server.py --root /home/www localhost 8080`	Idem, en utilisant un nom de domaine (localhost est défini dans /etc/hosts)
`python3 client.py --save /home/watch/small.model http://localhost:8080/llm/small 2> log.log`	Lance un client qui effectue une requête HTTP 0.9 au serveur se trouvant sur la même machine (`localhost`) sur le port 8080, et demande le fichier `llm/small` et l'enregistre dans le dossier `/home/watch/` sous le nom `small.model`. Tous les messages d'erreur et de log sont redirigés dans le fichier `log.log`.

2.5 Étapes conseillées pour réaliser le projet

Pour vous faciliter la réalisation du projet, nous vous proposons la liste des sous-tâches à effectuer.

1. Implémentation de l'encodage et du décodage des paquets (en particulier, faites attention à l'endianness des champs) dans une approche orientée tests. Des tâches INGInious sont prévue pour vous aider ; il est fortement conseillé de les réaliser.
2. Implémentation d'un serveur et d'un client simples sur UDP via l'API socket.
3. Fonctionnement du protocole SRTP avec un échange d'un seul paquet sur un réseau parfait.
4. Fonctionnement du protocole SRTP avec des échanges de n'importe quelle taille, dans le respect de toutes les spécifications, sur un lien présentant uniquement de la latence (par exemple, 200 ms), avec une suite de tests black-box vérifiant cela.
5. Fonctionnement du protocole SRTP sous toutes les conditions réseau possibles (pertes, corruption, troncation, jitter, latence) avec une suite de tests black-box vérifiant cela.
6. Évaluation des performances et identification des sources de limitations de votre implémentation.
7. Potentiellement, l'implémentation des acquittements sélectifs.

3 Tests

3.1 INGInious

Des tests INGInious seront fournis pour vous aider à tester deux facettes du projet :

1. La création, l'encodage et le décodage des paquets SRTP ;
2. L'envoi et la réception de données sur le réseaux, multiplexés sur un socket.

En d'autres termes, ces tests vérifieront que votre serveur sait envoyer un fichier de 1000 octets (1 bloc), et que votre client sait recevoir un fichier de 1000 octets (1 bloc) dans des conditions parfaites.

3.2 Tests individuels

Les tests INGInious ne suffiront pas à évaluer votre implémentation. Il vous est donc demandé de tester vous-même votre code, d'effectuer une suite de tests et de le documenter dans votre rapport. Cette suite de tests doit être automatisée à l'aide de `pytest`. Ces tests devront vérifier le bon fonctionnement du protocole en cas de pertes de paquets, de paquets mal formés, etc.

3.3 Tests d'interopérabilité

Vos programmes doivent être interopérables avec les implémentations d'autres groupes. Vous ne pouvez donc ni créer un nouveau type de segment ni ajouter des métadonnées au payload. Deux possibilités de tester l'interopérabilité de votre implémentation s'offrent à vous.

1. Nous mettrons à disposition un `server` et un `client` de référence sur Moodle (Compilés pour tournée sur les machines Didac). Ces exécutables utilitaires sont là pour vous aider. En revanche, ce ne sont pas ceux que nous utiliserons pour l'évaluation.
2. Une semaine avant la remise de la deuxième soumission, vous **devrez** tester votre implémentation avec 2 autres groupes (votre `python3 server.py` et leur `python3 client.py`, et votre `python3 client.py` et leur `python3 server.py`). Les résultats de ces tests d'interopérabilité devront figurer dans votre rapport final.

4 Planning et livrables

A. Première soumission, 23/03 à 23h59

1. Un jeu de slides (maximum 10 slides, au format PDF) décrivant l'architecture générale de votre programme et répondant **au minimum** aux questions suivantes : [4]
   - Que mettez-vous dans le champ Timestamp, et quelle utilisation en faites-vous ?
   - Comment avez-vous choisi la valeur du retransmission timeout ?
   - Quelle est la partie critique de votre implémentation, affectant la vitesse de transfert ?
   - Quelles sont les performances de votre protocole ? (**évaluation** graphique et/ou chiffrée, scénarios explorés, explication de la méthodologie)
   - Quelle(s) stratégie(s) de tests avez-vous utilisée(s) ?

4. Le rapport sera lu par des experts mandatés par l'entreprise qui connaissent le sujet. Soyez donc **objectifs** et évitez de réintroduire le contexte du projet dans votre rapport. Toutefois, libre à vous de mettre en avant une partie du projet non listée ci-dessous que vous trouvez pertinente.

— Si vous implémentez les acquittements sélectifs, voyez-vous des gains ? Si oui, lesquels ?

2. Implémentation des deux programmes permettant un transfert de données en utilisant le protocole.
3. Suite de tests des programmes et comment les exécuter.

B. Tests d'interopérabilité, durant la semaine du 23/03 au 27/03, salle Intel (planning à confirmer)

C. Deuxième soumission, 29/03 à 23h59

Mêmes critères que pour la première condition : les slides (maximum de 15 slides, au format PDF) doivent décrire en plus le résultat des tests d'interopérabilité, ainsi que les changements effectués au code, si applicable.

Format des livrables

La soumission fera en une seule archive **ZIP**, respectant le format suivant :

`/`	Racine de l'archive
`|- src/`	Le répertoire contenant le code source des deux programmes, donc au minimum `python3 server.py` et `python3 client.py`
`|- tests/`	Le répertoire contenant la suite de tests
`|- rapport.pdf`	Le rapport composé de slides, au format PDF
`|- gitlog.stat`	La sortie de la commande `git log --stat`

Cette archive sera nommée `projet1_nom1_nom2_nom3.zip`, où `nom1/2/3` sont les noms de famille des membres du groupe, et sera à mettre sur deux tâches dédiées sur Inginious (une pour chaque soumission) qui vérifieront le format de votre archive. Si celle-ci ne passe pas les tests de format, **votre soumission ne sera pas considérée pour l'évaluation !** Les liens vers ces tâches vous seront communiqués en temps utile.

Il est demandé de réaliser le projet avec un gestionnaire de versions (typiquement `git`).

☞ **Important :** l'évaluation tiendra compte de vos deux soumissions et pénalisera les projets dont les deux soumissions sont trop différentes. La première soumission n'est donc **PAS** facultative.

5 Evaluation

La note du projet sera composée des trois parties suivantes :

1. Implémentation : Vos programmes fonctionnent-ils correctement ? Qualité de la suite de tests ? Que se passe-t-il quand le réseau est non-idéal ? Sont-ils interopérables ? . . .Le respect des spécifications (arguments, fonctionnement du protocole, formats des segments, structure de l'archive, ...) est impératif pour la réussite de cette partie !
2. Rapport ;
3. Défense orale de votre projet

5.1 Défense orale

Nous vous demandons de venir défendre votre projet oralement la semaine suivant la deuxième soumission. L'objectif de cette défense orale est de nous assurer de la bonne compréhension du projet par tous les membres du groupe. Les modalités exactes de la présentation seront communiquées ultérieurement.

☞ **Important :** une absence injustifiée ou un échec à défendre votre projet oralement entraînera l'échec global du projet.

Il n'est donc pas utile de demander à un modèle d'intelligence artificielle de réaliser le projet à votre place et de soumettre cette implémentation si vous n'êtes pas capable de le faire vous-même.

Les oraux vont se dérouler durant la semaine 30 Mars au 03 Avril 2026. Les dates exactes vous seront communiquer ultérieurement sur Moodle.

6 Ressources utiles

6.1 Dissecteur Wireshark

Un dissecteur Wireshark vous permettant d'analyser vos traces et vous informant sur de potentielles erreurs dans le format de vos paquets est également fourni sur le moodle du cours. Pour l'installer, placez simplement le dissecteur (fichier .lua) dans le dossier plugin de Wireshark, dont l'emplacement peut être obtenu dans les paramètres de Wireshark (Help > About Wireshark > Folders > Personal Lua Plugins). Par défaut, le dissecteur décode uniquement les paquets SRTP envoyés sur le port `8080`. Décoder des paquets SRTP envoyés sur un autre port est également possible en sélectionnant SRTP dans l'option "Decode As..." apparaissant après avoir fait un clic droit sur un paquet.

6.2 Link Simulator

Nous mettons à votre disponibilité un outil pour simuler les pertes entre votre client et serveur. Vous pouvez télécharger l'outil via Moodle. Il contient un `README.md` qui vous explique comment l'utiliser.

Références

[1] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Dr. Allyn Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996. URL : https://www.rfc-editor.org/info/rfc2018, doi:10.17487/RFC2018.