

UNIVERSITE DE YAOUNDE I

ECOLE NATIONALE SUPERIEURE
POLYTECHNIQUE DE YAOUNDE

DEPARTEMENT DE GENIE
INFORMATIQUE



UNIVERSITY OF YAOUNDE I

NATIONAL ADVANCED SCHOOL
OF ENGINEERING OF YAOUNDE

DEPARTMENT OF COMPUTER
ENGINEERING

DÉVELOPPEMENT D'UN PROTOCOLE RÉSEAU
ÉCOÉNERGÉTIQUE BASÉ SUR QUIC POUR UN TRANSFERT
PARALLÈLE ET DISTRIBUÉ DE DONNÉES ENTRE SITES

MÉMOIRE DE FIN D'ÉTUDES

Présenté et soutenu par :

ANZIE SEVERIN BRADLEY

En vue de l'obtention du :

DIPLÔME D'INGÉNIEUR DE CONCEPTION, GÉNIE INFORMATIQUE

Sous la supervision de :

Pr. Thomas Djotio Ndié
Pr. Remous Aris Koutsiamanis

Devant le jury composé de :

Président : **Pettang Chrispin**, Professeur des Universités

Rapporteur : **Thomas Djotio**, Maitre de conférence UY1

Examineur : **Akam Denis**, Chargé de cours UY1

Invité : **Remous Aris Koutsiamanis**, Maitre de conférence IMT Atlantique Nantes

Année académique 2023-2024

Le 18 septembre 2024



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom



**DÉVELOPPEMENT D'UN PROTOCOLE RÉSEAU
ÉCOÉNERGÉTIQUE BASÉ SUR QUIC POUR UN
TRANSFERT PARALLÈLE ET DISTRIBUÉ DE
DONNÉES ENTRE SITES**

MÉMOIRE DE FIN D'ÉTUDES

Présenté et soutenu par :

ANZIE SEVERIN BRADLEY

En vue de l'obtention du :

DIPLOME D'INGÉNIEUR DE CONCEPTION, GÉNIE INFORMATIQUE

Année académique 2023-2024

Le 22 septembre 2024

DÉDICACE



REMERCIEMENTS

Il est difficile de trouver les mots justes pour exprimer ma profonde gratitude envers toutes les personnes qui ont contribué à mon éducation et à la réalisation de ce mémoire. Ce projet n'aurait pas été possible sans le soutien inestimable de ceux qui m'entourent.

Je tiens à remercier chaleureusement mes encadreurs de stage, Mcf. **Remous Aris Koutiamanis**, Pr. **Jean-Marc Menaud** et Pr **Thomas Djotio Ndié**, pour leur accompagnement exceptionnel, leur patience et leurs conseils. Un merci particulier à **Aris**, dont le soutien indéfectible et la passion pour la recherche ont été des sources d'inspiration. Je suis également reconnaissant au Pr **Bouetou Bouetou Thomas**, chef du département de Génie Informatique, pour son écoute et son engagement envers ses étudiants.

Je remercie aussi l'ensemble des enseignants et du personnel pédagogique de notre département. Leur enseignement de qualité m'a permis d'acquérir des connaissances solides et d'affiner ma réflexion, essentielles à l'élaboration de ce travail.

Un immense merci à mes camarades de la promotion **GI2024**. Votre camaraderie et votre soutien mutuel ont rendu cette expérience mémorable. Ensemble, nous avons partagé des moments précieux qui resteront gravés dans ma mémoire.

Je souhaite aussi remercier mes amis et camarades de stage, **Menra Romial** et **Guimapi Céleste**, pour les moments passés et pour leur soutien tout au long de cette période. Merci également aux ingénieurs **Arnaud TAMO** et **Gabin Fodop** pour leurs conseils constructifs et leur aide précieuse durant mon stage. Un remerciement spécial à mon ami **Marc Djiala**, pour son soutien, son attention et les moments partagés.

Je tiens à exprimer une reconnaissance particulière à mon amie **Mekena Viannie** pour son soutien inconditionnel et ses conseils avisés, qui ont été d'une grande aide tout au long de ce projet. A mes **Mojañ** qui sont avec moi dans ce parcours depuis le début et avec qui j'ai partagé l'essentiel de mon temps, merci à vous mes frères.

Je ne saurais oublier ma famille, ma plus grande source de force et de motivation. Merci à ma mère, **Mine'e Mi Oba**, pour son amour inconditionnel ; à ma grand-mère, **Anzie Rose**, et à mon grand-père, **Oba Olle Théophile**, pour leur sagesse et leurs précieux conseils ; ainsi qu'à mon oncle, **Oyono Oba Peter**, pour son soutien. Une mention spéciale à mes frères et sœurs, dont le soutien a été inestimable. Votre foi en moi a été essentielle dans ma réussite.

Enfin, je dédie ce travail à toutes les personnes passionnées par la technologie et l'innovation,



qui m'ont inspiré à explorer de nouveaux horizons. Ce mémoire est le fruit de nombreuses influences et de l'engagement d'une communauté qui m'a soutenu à chaque étape.

Merci à tous pour votre précieuse contribution à ce projet.




TABLE DES MATIÈRES

Dedication	i
Acknowledgements	ii
Abstract	vi
Résumé	vii
Tableaux	viii
Figures	ix
Table of Contents	xi
Dedication	i
Remerciements	ii
Sigles et abréviations	vii
Glossaire	viii
Résumé	x
Abstract	xi
Liste des tableaux	xii
Table des figures	xiii
Introduction Générale	1
1 Concepts généraux et État de l'art	4
1.1 Généralité	5

1.1.1	Définition et Concepts de Base	5
1.1.2	Rappels sur les Protocoles de Transfert de Données	5
1.2	Présentation du protocole QUIC	10
1.2.1	introduction	10
1.2.2	Principe de fonctionnement	11
1.2.3	Les principaux concepts autour du protocole QUIC	12
1.2.4	Extension de QUIC pour le multipath : MP-QUIC	14
1.2.5	Etude comparative des protocoles de transport QUIC, TCP et UDP	15
1.3	Dimension Énergétique dans les Protocoles de Transport Décentralisés	16
1.3.1	Vers des Protocoles Écoénergétiques	17
1.4	État de l'art	17
1.4.1	Étude des Protocoles Existants	17
1.4.2	HTTP	17
1.4.3	FTP	18
1.4.4	SFTP	19
1.4.5	GridFTP	20
1.4.6	BitTorrent	20
1.4.7	FDT	21
1.4.8	MDTMFTP	22
1.4.9	Limites des Protocoles Existants	22
1.4.10	Impact du Protocole QUIC	23
1.4.11	Étude Comparative des protocoles de tranfert de données existant	23
1.5	Bilan du chapitre et Positionnement	25
1.5.1	Bilan	25
1.5.2	Positionnement	25
2	Conception du protocole DataStreamX	26
2.1	Rappel du problème et des objectifs poursuivis	27
2.1.1	Objectif général	27
2.1.2	Modélisation mathématique du problème d'optimisation	27
2.1.3	Contraintes du problème	28
2.1.4	Complexité du problème	29
2.2	Modélisation UML du protocole DataStreamX	30
2.2.1	Spécification des exigences	30
2.2.2	Analyse	31
2.2.3	Conception détaillée	43
2.2.4	Simulation et Validation	50
2.3	Algorithmes d'orchestration du transfert des données	51
2.3.1	Calcul des méta-paramètres statiques	52

TABLE OF CONTENTS



2.3.2	Sélection des relais	52
2.3.3	Répartition optimale des segments	54
3	Implémentation et résultats	56
3.1	Choix des technologies et des outils	57
3.1.1	Les langages de programmation et les bibliothèques	57
3.1.2	Les outils utilisés	57
3.2	Implémentation du Protocole	59
3.2.1	Architecture Modulaire	59
3.2.2	Gestion des Connexions Multiplexées	60
3.2.3	Calcul Adaptatif des Fenêtres et des Segments	60
3.2.4	Répartition Dynamique des Segments de Données	60
3.2.5	Gestion des Paquets	61
3.2.6	Aspect énergétique	61
3.3	Environnement de simulation	62
3.3.1	Le serveur	62
3.3.2	Les tâche	64
3.3.3	Le réseau	64
3.4	Résultats des test et simulation du Protocole	64
3.4.1	Configuration du Benchmark	64
3.4.2	Indicateurs de Performance	65
3.4.3	Résultats du Benchmark	65
3.4.4	Comparaison par rapport aux autres protocoles existants	66
3.4.5	Analyse des Résultats	66
	Conclusion Générale et Perspectives	69
	Références bibliographiques	71



SIGLES ET ABRÉVIATIONS

FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IMT	Institut Mines et Télécommunications
IA	Intelligence Artificielle
IOT	Internet Of Things
IP	Internet protocol
ISO	International Organization for Standardization
ENSPY	Ecole Nationale Supérieure Polytechnique de Yaoundé
FDT	Fast Data Transfer
MDTMFTP	Modified Data Transfer Model File Transfer Protocol
NIC	Network Interface Card
QUIC	Quick UDP Internet Connections
RTT	Round-trip Time
SFTP	Secure Shell File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocols

GLOSSAIRE

ACK (Acknowledgment) Message envoyé par le récepteur d'un paquet pour confirmer la bonne réception de celui-ci. Utilisé dans les protocoles de communication pour garantir la fiabilité des transmissions.

Bande passante Quantité de données qui peuvent être transmises sur une connexion réseau dans un temps donné. Elle est généralement exprimée en bits par seconde (bps).

Buffer Zone de mémoire utilisée pour stocker temporairement les données en transit entre deux entités, par exemple, entre un client et un serveur. Les buffers permettent de compenser les différences de vitesse de traitement entre l'émetteur et le récepteur.

Client Dans le contexte du protocole, il s'agit de l'entité qui initie une connexion avec un serveur pour demander ou recevoir des données.

Connexions Multiplexées Technique permettant de gérer plusieurs flux de données sur une même connexion réseau. Elle optimise l'utilisation de la bande passante en réduisant les temps d'attente et en augmentant la flexibilité des échanges.

Contrôle de Flux (Flow Control) Mécanisme utilisé pour gérer le taux de transmission des données entre deux entités afin de prévenir la surcharge des buffers et éviter la perte de paquets.

Contrôle de Congestion Méthode permettant de réguler l'envoi de données sur un réseau afin de prévenir et de gérer la congestion. Il ajuste dynamiquement le débit de transmission en fonction des conditions du réseau.

Fenêtre de Transfert Segment d'un fichier déterminé pour être envoyé en un seul bloc. Le protocole décrit divise les fichiers en plusieurs fenêtres, chaque fenêtre étant ensuite segmentée pour différents relais.

MP-QUIC Variante du protocole QUIC permettant d'utiliser plusieurs chemins de réseau simultanément pour le transfert de données. Améliore la résilience et la performance du protocole en multipliant les routes possibles pour les paquets.

Packet (Paquet) Unité de données transmise sur un réseau. Un paquet contient généralement une en-tête (avec des informations de contrôle) et une charge utile (les données elles-mêmes).



Paquet de Contrôle Type de paquet utilisé pour gérer la communication et la synchronisation entre les entités du protocole. Peut inclure des informations telles que des identifiants de connexion ou des notifications de fin de session.

Paquet de Données Paquet contenant la charge utile de la communication, c'est-à-dire les données effectivement transférées entre le client, le serveur, et les relais.

Protocole de Transport Ensemble de règles et de conventions qui régissent l'envoi et la réception de données entre deux entités communicantes. QUIC est un exemple de protocole de transport utilisé dans le protocole décrit ici.

QUIC Protocole de transport orienté connexion, conçu pour être plus performant que TCP en réduisant la latence et en intégrant le chiffrement dès le départ. Utilisé comme base pour le protocole décrit.

QUIC-go Implémentation du protocole QUIC en langage Go. Fournit une infrastructure pour le développement et le test de protocoles de transport basés sur QUIC.

Relais Entité intermédiaire dans le protocole décrit qui reçoit des segments de données du serveur central et les transmet au client. Le relais aide à répartir la charge et à optimiser le transfert des données.

Round-Trip Time (RTT) Temps total nécessaire pour qu'un signal soit envoyé et qu'un accusé de réception revienne. Mesure essentielle pour évaluer la performance des connexions réseau.

Segment Partie d'une fenêtre de transfert assignée à un relais spécifique. Les segments sont définis en fonction des performances de chaque relais pour optimiser le transfert des données.

Sérialisation/Désérialisation Processus de conversion des données en une forme qui peut être stockée ou transmise (sérialisation) et de conversion inverse pour les rendre exploitables (désérialisation). .



RÉSUMÉ

Les transferts de données à grande échelle entre centres de données posent des défis majeurs, notamment en termes de performances et de consommation énergétique. **DataStreamX** propose une approche décentralisée qui répartit les flux sur plusieurs points de connexion pour éviter les limitations liées à la centralisation des transferts. Grâce à une architecture distribuée et à des algorithmes adaptatifs, le protocole optimise la bande passante disponible et assure un débit stable même en période de forte demande. En parallèle, DataStreamX se distingue par son engagement écologique, en favorisant l'utilisation de relais alimentés par des énergies renouvelables pour réduire l'empreinte carbone des transferts. Les résultats expérimentaux démontrent une amélioration des performances de transfert tout en réduisant la consommation énergétique par rapport aux approches classiques, ce qui fait de DataStreamX une solution innovante pour les besoins croissants des infrastructures numériques.

Mots clés : Protocole de transfert de données, Protocole QUIC, Décentralisation, Optimisation énergétique, Architecture distribuée.

ABSTRACT

Large-scale data transfers between data centers face significant challenges, particularly regarding performance and energy consumption. **DataStreamX** offers a decentralized approach that distributes data flows across multiple connection points to overcome the limitations of centralized transfers. Through a distributed architecture and adaptive algorithms, the protocol optimizes available bandwidth and maintains stable throughput even during high-demand periods. Additionally, DataStreamX stands out for its ecological commitment, prioritizing the use of renewable energy-powered relays to reduce the carbon footprint of data transfers. Experimental results show improved transfer performance while reducing energy consumption compared to traditional methods, positioning DataStreamX as an innovative solution for the growing needs of digital infrastructures.

Keywords : Data transfer protocol, QUIC Protocol, Decentralization, Energy optimization, Distributed architecture

LISTE DES TABLEAUX

1.1	Tableau comparatif des protocoles QUIC, TCP et UDP	16
1.2	Comparaison des protocoles de transfert de données existant à QuicPlex	24
3.1	Résultats du Benchmark	66
3.2	Comparaison des protocoles de transfert de données existant à QuicPlex	67

TABLE DES FIGURES

1.1	Architecture client-serveur vs architecture P2P	6
1.2	QUIC architecture.[4]	11
1.3	QUIC Connection establishment 0-RTT and 1-RTT handshake.[4]	12
1.4	QUIC stream flow control.[4]	14
1.5	Processus de communication HTTP.[18]	18
1.6	Fonctionnement modes actif et passif du protocole FTP.[23]	19
2.1	<i>Diagramme d'activité global du protocole DataStreamX</i>	32
2.2	<i>Diagramme de séquence DataStreamX</i>	34
2.3	<i>Diagramme de séquence Ouverture de session</i>	35
2.4	<i>Diagramme de séquence Refus de session</i>	36
2.5	<i>Diagramme de séquence Acceptation de session</i>	37
2.6	<i>Diagramme de séquence envoi de données</i>	38
2.7	<i>Liste des signaux échangée dans DataStreamX</i>	39
2.8	<i>Interfaces des signaux échangés dans DataStreamX</i>	40
2.9	<i>Structures de Composant des entités de base de DataStreamX</i>	40
2.10	<i>Structure interne des composants DataStreamX</i>	41
2.11	<i>Structures de Composant des entités de base de DataStreamX</i>	42
2.12	<i>Architecture fonctionnelle du protocole DataStreamX</i>	43
2.13	<i>Structures d'un échange entre deux composants DataStreamX</i>	44
2.14	<i>Ouverture de session</i>	46
2.15	<i>Refus de session</i>	47
2.16	<i>Acceptation de session</i>	47
2.17	<i>Envoi de données</i>	48
2.18	<i>Structures d'un paquet DataStreamX</i>	49
2.19	<i>Simulation et validation de DataStreamX</i>	50
2.20	<i>Simulation et validation de DataStreamX (suite)</i>	51

INTRODUCTION GÉNÉRALE

CONTEXTE

Le transfert de grands volumes de données entre centres de données, dans des secteurs tels que le Cloud Computing, l'Internet des Objets (IoT), le Big Data et l'Intelligence Artificielle (IA), représente un défi considérable. L'augmentation exponentielle des échanges de données, associée à la nécessité croissante de rapidité et d'efficacité, soulève des préoccupations majeures en termes de performance et d'impact environnemental. Bien que les avancées en matière de bande passante permettent désormais d'atteindre des débits de plusieurs centaines de Gb/s, la centralisation des transferts de données dans les infrastructures actuelles continue de limiter l'efficacité globale. Cette centralisation impose une récupération des données, souvent stockées sur des disques distribués, vers un point unique avant leur transfert. Ce processus crée des limitations structurelles qui ralentissent les flux de données et réduisent l'efficacité des opérations, en particulier lors de transferts massifs.

Parallèlement, les questions environnementales ont pris une importance cruciale. L'optimisation de la consommation énergétique des infrastructures numériques est désormais indispensable, surtout à mesure que les centres de données se multiplient et consomment d'énormes quantités d'énergie. La transition vers l'utilisation de sources d'énergie renouvelables et l'optimisation des infrastructures pour minimiser l'empreinte carbone deviennent des priorités dans la conception des solutions de transfert de données. Il est donc impératif de développer des protocoles qui non seulement améliorent les performances de transfert, mais qui répondent également aux exigences d'efficacité énergétique pour une meilleure durabilité.

Face à ces défis, il est nécessaire de repenser les protocoles actuels, en particulier ceux basés sur la centralisation, et d'adopter des approches décentralisées capables de maximiser la bande passante tout en optimisant l'utilisation des énergies renouvelables. C'est dans ce contexte que DataStreamX propose un protocole distribué, visant à surmonter ces obstacles et à ouvrir la voie à des transferts de données plus performants et respectueux de l'environnement.



PROBLÉMATIQUE

Malgré les progrès substantiels dans les technologies réseau, notamment en matière de bande passante et d'interconnexion des centres de données, la centralisation des transferts de données constitue toujours un obstacle majeur à l'optimisation des performances et de l'efficacité énergétique. La centralisation limite les débits, crée des points de congestion et accroît la consommation énergétique des infrastructures. Cela soulève plusieurs questions critiques :

- **Comment concevoir un protocole de transfert de données distribué et parallèle capable de surmonter les limitations structurelles liées à cette centralisation ?**
- **Comment maximiser l'efficacité énergétique des transferts tout en assurant des performances élevées, notamment en termes de débits ?**
- **Comment optimiser la répartition des flux de données en prenant en compte l'impact énergétique des relais, en favorisant l'utilisation de sources d'énergie renouvelables ?**

Ces problématiques sont au cœur des infrastructures numériques modernes. Elles nécessitent des solutions qui concilient la performance des transferts de données avec une gestion efficace de l'énergie, dans un cadre où la demande de rapidité et d'évolutivité est croissante, mais où l'impact environnemental doit être minimisé.

OBJECTIF

L'objectif principal de ce travail est de développer un protocole de transfert de données distribué, basé sur le protocole QUIC, permettant de surmonter les limitations liées à la centralisation des transferts et d'améliorer l'efficacité énergétique. Ce protocole vise à maximiser les performances de transfert tout en minimisant l'empreinte carbone des infrastructures en favorisant l'utilisation de relais alimentés par des sources d'énergie renouvelables. Pour atteindre cet objectif global, trois sous-objectifs guideront notre démarche :

- 🍃 **La décentralisation des transferts de données :** Concevoir un protocole de transfert de données décentralisé qui évite les goulots d'étranglement liés à la centralisation, permettant un transfert parallèle, rapide et fiable des données. Il s'agira de développer une répartition intelligente des segments de fichiers entre différents relais en fonction de leurs capacités réseau.
- 🍃 **L'optimisation dynamique des flux de données :** Développer et mettre en œuvre des algorithmes de synchronisation des flux de données capables de s'adapter aux fluctuations de la bande passante. Ces algorithmes devront coordonner les interfaces réseau des relais et des clients pour garantir une transmission fluide même en présence de variations dans la disponibilité des ressources.





- ☛ **La sélection énergétique des relais :** Mettre en place des mécanismes de sélection des relais en fonction de leur consommation énergétique et de leur source d'énergie. L'objectif est de favoriser les relais utilisant des sources d'énergie renouvelables, dans le but de réduire l'empreinte carbone tout en maintenant des performances de transfert élevées.

PLAN DU MÉMOIRE

Ce mémoire est structuré en trois principaux chapitres :

- ☛ **Chapitre 1** *Généralités et État de l'art* : Dans premier ce chapitre, nous présentons les concepts fondamentaux des protocoles de transfert de données, nécessaires à la compréhension des enjeux liés à la centralisation et de la solution proposée. Une introduction détaillée au protocole QUIC et à la notion de multipath sera fournie, suivie d'une étude fonctionnelle des protocoles de transfert existants.
- ☛ **Chapitre 2** *Conception du protocole DataStreamX* : Ce chapitre est dédié à la conception du protocole. Nous y développerons un modèle mathématique permettant de résoudre le problème d'optimisation lié aux transferts décentralisés. Une modélisation détaillée du protocole à l'aide du langage UML sera proposée, accompagnée d'une description des différents algorithmes utilisés pour coordonner les transferts parallèles et garantir une répartition efficace des flux de données entre les relais. L'optimisation énergétique et la gestion des connexions y seront également abordées.
- ☛ **Chapitre 3** *Implémentation et évaluation résultats* : Ce dernier chapitre détaille l'implémentation du protocole DataStreamX. Nous y présentons les technologies choisies, l'environnement de développement et de test, ainsi que la méthodologie employée. Les résultats expérimentaux obtenus seront discutés, notamment en termes de performances de transfert et de réduction de la consommation énergétique, comparés aux méthodes classiques.

Nous terminerons par une conclusion générale, qui fait un bilan de ce travail, ressort ses limites et fournit quelques perspectives.



CONCEPTS GÉNÉRAUX ET ÉTAT DE L'ART

Dans ce premier chapitre, nous présentons les concepts de base nécessaires à la compréhension du protocole proposé. Nous aborderons les fondamentaux des **protocoles de communication**, en particulier les caractéristiques du **protocole QUIC** et son importance dans l'amélioration des performances des transferts de données. Les notions de **débit**, de **latence** et de **multipath** seront également détaillées. Nous réaliserons une analyse des principaux **protocoles de transfert de données** actuellement utilisés, en mettant l'accent sur leurs forces et limitations, et nous concluons par une **étude comparative**.



1.1 Généralité

1.1.1 Définition et Concepts de Base

Les **protocoles** réseau sont les ensembles de **règles** et de **conventions** qui permettent la communication entre différents dispositifs au sein d'un réseau. Ils définissent la manière dont les données sont envoyées, reçues, et interprétées sur le réseau. Les protocoles réseau se classifient en différentes couches selon le modèle OSI (Open Systems Interconnection) ou le modèle TCP/IP, qui sont des frameworks utilisés pour comprendre et concevoir des systèmes de communication en réseau.

Les protocoles de transport sont responsables de la livraison fiable et ordonnée des données entre les applications qui communiquent sur un réseau. Ils assurent la gestion des flux de données, le contrôle des erreurs, et la gestion de la congestion. Les deux principaux protocoles de transport sont

1.1.2 Rappels sur les Protocoles de Transfert de Données

1.1.2.1 Architectures des Protocoles de Transfert de Fichiers

Les architectures de protocoles de transfert de fichiers sont variées, mais elles partagent un objectif commun : optimiser le transfert de données entre un ou plusieurs nœuds dans un réseau. Ces architectures peuvent être centralisées ou décentralisées. Les systèmes centralisés reposent sur un serveur unique qui gère les requêtes de transfert, tandis que les systèmes décentralisés, comme BitTorrent, permettent une distribution des tâches de transfert entre plusieurs pairs, améliorant ainsi l'efficacité et la résilience. Les quatre architectures les plus courantes sont les suivantes :

Architecture Client-Serveur

L'architecture client-serveur est l'une des plus classiques et largement utilisées pour les transferts de données. Dans ce modèle, un serveur centralisé héberge les données et gère les requêtes des clients, qui se connectent pour télécharger ou envoyer des fichiers. Dans cette architecture, on retrouve des protocoles tels que FTP, HTTP/HTTPS et GridFTP. Cette architecture présente des avantages parmi lesquels :

- **La centralisation** qui permet un contrôle strict de l'accès aux données et une gestion simplifiée.
- **La facilité** de mise en œuvre et de gestion pour les administrateurs.

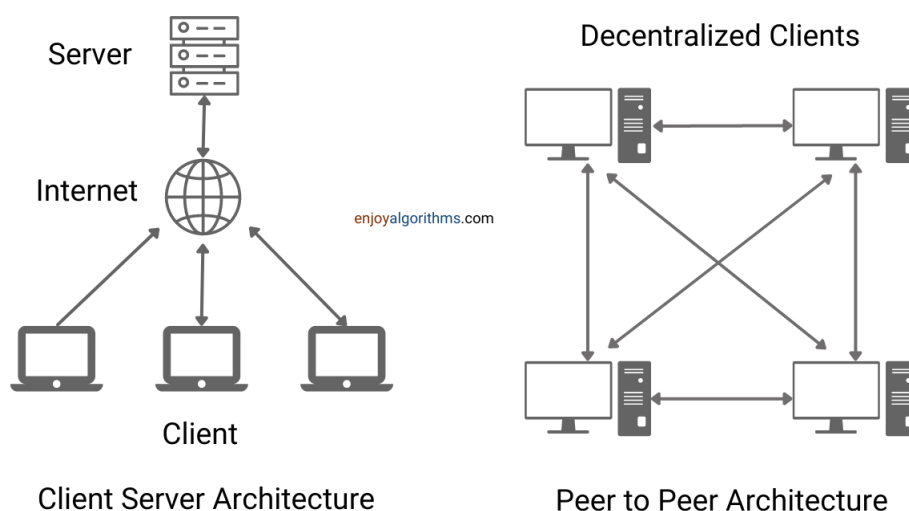
L'architecture client-serveur, bien que simple et intuitive, présente néanmoins des limitations sévères, à savoir :



- les **goulots d'étranglement** possibles au niveau du serveur.
- La **vulnérabilité** accrue en cas de défaillance du serveur central.

Architecture Peer-to-Peer (P2P)

FIGURE 1.1 – Architecture client-serveur vs architecture P2P



L'architecture Peer-to-Peer (P2P) est une architecture décentralisée, permettant à chaque nœud du réseau d'agir à la fois comme client et serveur. Les données sont partagées directement entre les pairs, sans besoin de serveur central. Par rapport à l'architecture client-serveur, cette architecture permet de résoudre la vulnérabilité due à la présence d'un seul serveur pouvant paralyser le transfert en cas de panne. Deux principaux protocoles de transfert de données utilisent cette architecture **BitTorrent**[3], Un protocole populaire qui divise les fichiers en morceaux, lesquels sont distribués entre les utilisateurs qui partagent et téléchargent simultanément et [?] Utilisé dans les premières applications de partage de fichiers, où les utilisateurs se connectaient directement pour partager des fichiers. Parmi ses avantages, on distingue entre autres :

- **Résilience** accrue grâce à l'absence d'un point unique de défaillance.
- **Haute scalabilité**, chaque nouveau pair augmentant la capacité totale du réseau.

malgré ces avantages, cette architecture présente des limitations notoires :

- **Complexité** accrue dans la gestion et le suivi des ressources.
- **Problèmes potentiels de sécurité et de fiabilité** des données.



Architecture Hybride (Client-Serveur + P2P)

L'architecture hybride combine les avantages des architectures client-serveur et P2P. Un serveur central peut gérer la coordination ou l'indexation, tandis que le transfert réel des données se fait directement entre les pairs. On distingue deux principaux protocoles dans cette architecture à savoir **eMule** qui utilise un serveur central pour l'indexation des fichiers, mais les fichiers eux-mêmes sont transférés entre les utilisateurs via P2P et **Skype** qui dans ses versions antérieures utilisait une architecture hybride où les serveurs centraux géraient les connexions, mais les appels audio/vidéo étaient transférés directement entre les utilisateurs. En termes d'avantages, cette approche assure

- **Une meilleure gestion des ressources** grâce à la coordination centralisée.
- **Une scalabilité améliorée** tout en maintenant un certain niveau de contrôle centralisé.

parmi ses inconvénients, on retrouve :

- **Complexité** plus élevée en raison de la gestion des deux types de connexions.
- **Vulnérabilité** partielle au niveau du serveur central en cas de défaillance.

Architecture Multi-Source/Segmented Download

Cette architecture permet de télécharger simultanément des segments d'un fichier à partir de multiples sources, ce qui peut considérablement augmenter la vitesse de téléchargement. Elle est souvent utilisée dans des environnements distribués ou avec des systèmes de mise en cache. Comme protocole implémentant cette architecture, on retrouve **HTTP avec CDN** bien que HTTP soit traditionnellement client-serveur, les CDN utilisent une architecture multi-source pour servir des segments de contenu à partir de serveurs répartis géographiquement et **MDTMFTP** qui permet de transférer des fichiers en parallèle à partir de plusieurs sources, optimisant ainsi la bande passante disponible. cette architecture présente des avantages significatifs à savoir :

- Amélioration significative des performances de téléchargement.
- Résilience accrue en cas de défaillance d'une source.

mais aussi des difficultés considérables :

- **Complexité** dans la reconstitution des fichiers à partir de segments provenant de sources multiples.
- **Potentielle incohérence** des données si les sources ne sont pas bien synchronisées.

1.1.2.2 Les Protocoles de Transport comme Substrat

Les protocoles de transfert de données permettent de coordonner la progression du transfert des données, mais pour assurer la livraison des données d'un point à un autre, ils se basent sur des protocoles de transport.





Les protocoles de transport forment la couche critique sur laquelle reposent les transferts de données sur Internet. Ils déterminent la manière dont les données sont segmentées, envoyées, et réassemblées à destination, tout en gérant la fiabilité, la congestion, et la latence. Les deux principaux protocoles de transport, TCP et UDP, offrent différentes approches avec des avantages et des inconvénients spécifiques pour le transfert de données.

1.1.2.3 TCP

TCP est un protocole de transport orienté connexion, ce qui signifie qu'il établit une connexion fiable entre l'émetteur et le récepteur avant de commencer le transfert de données. Ce protocole assure l'ordonnancement et l'intégrité des paquets de données en utilisant des mécanismes de contrôle de flux, de détection et de correction d'erreurs. Parmi les principaux avantages qu'offre TCP on retrouve :

- **Fiabilité** : TCP garantit que toutes les données sont livrées, dans l'ordre, et sans duplication. Si des paquets sont perdus, TCP les retransmet automatiquement.
- **Contrôle de flux** : TCP ajuste dynamiquement le débit d'envoi pour éviter la congestion réseau, ce qui aide à prévenir les surcharges et à optimiser l'utilisation de la bande passante.
- **Large adoption** : En raison de sa fiabilité, TCP est le protocole de transport le plus utilisé pour des applications nécessitant une transmission sécurisée des données, comme HTTP/HTTPS, FTP, et l'email.

Ce protocole présente des inconvénients qui peuvent fortement ralentir un protocole de transfert des données, notamment :

- **Latence élevée** : L'établissement de la connexion (handshake à trois voies) et les mécanismes de retransmission peuvent introduire une latence importante, surtout dans les réseaux avec des délais de propagation élevés.
- **Surcharge de bande passante** : Les mécanismes de contrôle de flux et de correction d'erreurs ajoutent une surcharge de bande passante, ce qui peut être inefficace pour les petites transmissions ou les réseaux à faible bande passante.
- **Scalabilité limitée** : TCP est moins adapté aux environnements où une scalabilité massive est requise, comme dans le cas des architectures décentralisées avec de nombreux nœuds.

1.1.2.4 UDP

UDP est un protocole de transport sans connexion qui envoie les données sous forme de datagrammes, sans vérifier si elles sont correctement arrivées à destination. Il n'y a pas d'établissement de connexion préalable, ce qui le rend beaucoup plus rapide que TCP pour l'envoi de





petits volumes de données ou dans des situations où la latence est critique. cette simplicité du protocole UP lui accorde un certain nombre d'avantages par rapport à TCP tels que

- **Faible latence** : Sans les procédures d'établissement de connexion et de retransmission, UDP minimise la latence, ce qui le rend idéal pour les applications nécessitant une transmission rapide, comme le streaming vidéo, les jeux en ligne, et la VoIP.
- **Simplicité** : L'absence de mécanismes complexes (comme le contrôle de flux) réduit la surcharge, ce qui le rend plus efficace pour les transmissions où la fiabilité n'est pas critique.
- **Flexibilité** : UDP permet aux développeurs de gérer manuellement la fiabilité et la correction d'erreurs, ce qui est avantageux dans les systèmes où le contrôle fin du transfert de données est nécessaire.

c'est également cette simplicité du protocole UDP qui fait ses principales faiblesses

- **Aucune garantie de livraison** : UDP n'offre aucune garantie que les paquets arrivent à destination ni qu'ils soient dans le bon ordre, ce qui peut entraîner une perte de données ou une désynchronisation.
- **Absence de contrôle de congestion** : Sans mécanismes intégrés de gestion de la congestion, UDP peut aggraver les problèmes de surcharge réseau, surtout dans les environnements très sollicités.
- **Risques de sécurité** : La simplicité de UDP le rend vulnérable à des attaques comme le spoofing et les inondations, ce qui nécessite des mesures de sécurité supplémentaires lors de son utilisation.

1.1.2.5 Notion de Multi-chemin et Protocoles de transport Associés

Le concept de multi-chemin (ou multipath) consiste à permettre à une connexion réseau d'utiliser plusieurs chemins simultanément pour le transfert de données. Cela permet d'améliorer plusieurs aspects du transfert, notamment la résilience, la performance et l'équilibrage de charge. Les protocoles basés sur le multi-chemin sont particulièrement adaptés aux environnements où la connectivité peut être variable, comme les réseaux mobiles, les environnements de cloud, ou les architectures décentralisées. Parmi ces protocoles on retrouve principalement

MPTCP

MPTCP est une extension du protocole TCP qui permet à une seule connexion TCP d'utiliser plusieurs chemins simultanément. Il a été développé pour surmonter les limitations de TCP dans des environnements où la connectivité et la bande passante peuvent varier, comme dans les réseaux mobiles ou les architectures multi-homed (dispositifs ayant plusieurs interfaces réseau). Il fonctionne en subdivisant une connexion TCP en plusieurs sous-flux, chacun étant capable d'utiliser un chemin réseau différent. Cela permet d'optimiser l'utilisation des ressources réseau disponibles, en utilisant la capacité totale de plusieurs liens.





SCTP

SCTP est un protocole de transport orienté connexion, conçu à l'origine pour le transport de signaux de télécommunication sur IP, mais qui a trouvé d'autres applications dans des environnements nécessitant une transmission fiable de données en flux multiple. Il permet la transmission de plusieurs flux indépendants de données entre deux hôtes, chacun pouvant être transmis sur un chemin différent. SCTP intègre également des mécanismes de gestion de la congestion et de tolérance aux pannes, similaires à ceux de TCP. ces deux protocoles présentent une avancée considérable en terme de protocoles de transport s'accompagnant ainsi de plusieurs avantages

- **Multiplexage de flux** : SCTP permet à plusieurs flux indépendants de données d'être transmis simultanément sur une même connexion, évitant ainsi les blocages tête de ligne qui peuvent survenir avec TCP.
- **Tolérance aux pannes** : SCTP est capable de détecter une défaillance de chemin et de rerouter les flux de données vers un autre chemin, similaire à MPTCP.
- **Sécurité accrue** : SCTP intègre des mécanismes de protection contre les attaques de type SYN-flooding, plus robustes que ceux de TCP.

et des inconvénients inhérents :

- **Adoption limitée** : SCTP est moins largement déployé que TCP et UDP, ce qui limite son utilisation dans les environnements réseau courants.
- **Complexité** : Comme MPTCP, SCTP est plus complexe à implémenter et à gérer que TCP ou UDP, ce qui peut compliquer son adoption et son déploiement.

Bien que performants pour le transport des données, ces protocoles souffrent des diverses difficultés d'adoption et de mise en œuvre. C'est par exemple le cas de TCP qui est fortement couplé au noyau du système d'exploitation hôte[19].

1.2 Présentation du protocole QUIC

1.2.1 introduction

QUIC est un protocole de transport initié par Google en 2012, avec une première version officielle introduite par l'Internet Engineering Task Force (IETF) en 2021. Il a été développé pour résoudre plusieurs problèmes associés au protocole TCP. Premièrement, dans TCP, la perte d'un paquet empêche tous les paquets suivants d'atteindre l'application en raison de la livraison séquentielle et de la gestion des flux basée sur les fenêtres. Deuxièmement, TCP utilise l'adresse IP et le numéro de port pour identifier une connexion, ce qui nécessite une négociation coûteuse en cas de changement d'adresse IP ou de numéro de port, car il ne permet pas la réutilisation du contexte de communication. Troisièmement, TCP ne prend pas en charge nativement le multiplexage. Plutôt que d'améliorer TCP, il a été décidé de développer un nouveau protocole.

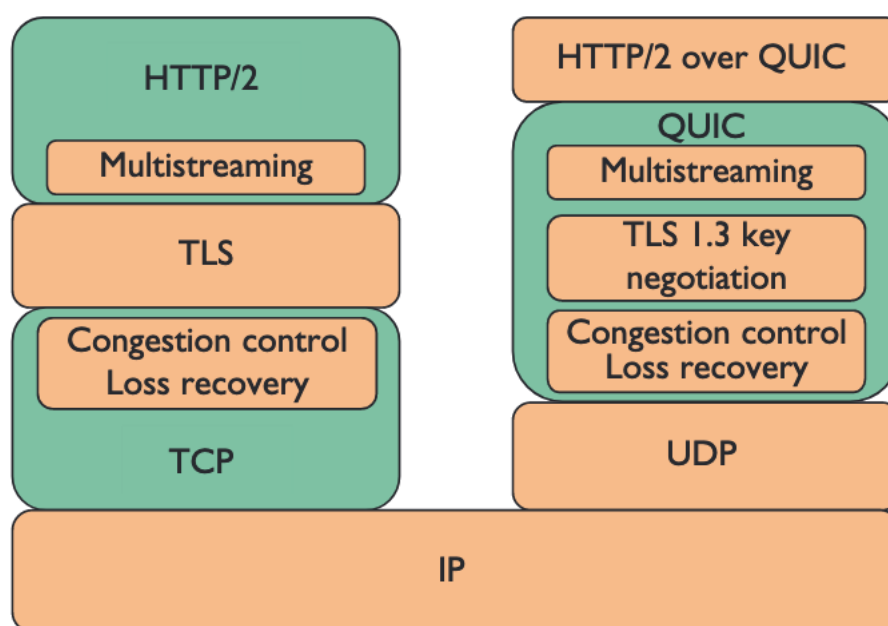




En effet, TCP est implémenté dans le noyau, alors que QUIC est implémenté sur UDP, ce qui le rend plus flexible. De plus, QUIC intègre des fonctionnalités telles que le multiplexage, la cryptographie, le contrôle de congestion et la récupération des pertes, contrairement à TCP où ces fonctionnalités sont fixes [4, 19].

1.2.2 Principe de fonctionnement

FIGURE 1.2 – QUIC architecture.[4]



QUIC fonctionne en remplaçant la majorité de la pile HTTPS traditionnelle, combinant les fonctionnalités de HTTP/2, TLS et TCP. Il utilise UDP comme substrat, permettant à ses paquets de traverser les middleboxes (dispositifs intermédiaires) tout en maintenant une sécurité et une intégrité des données élevées grâce à un chiffrement et une authentification intégrée. Cette utilisation d'UDP permet également une plus grande flexibilité et rapidité dans la mise à jour et le déploiement du protocole. Pour mieux comprendre le fonctionnement de QUIC, examinons en détail ses principales composantes et la manière dont elles améliorent les performances et la sécurité des communications réseau.



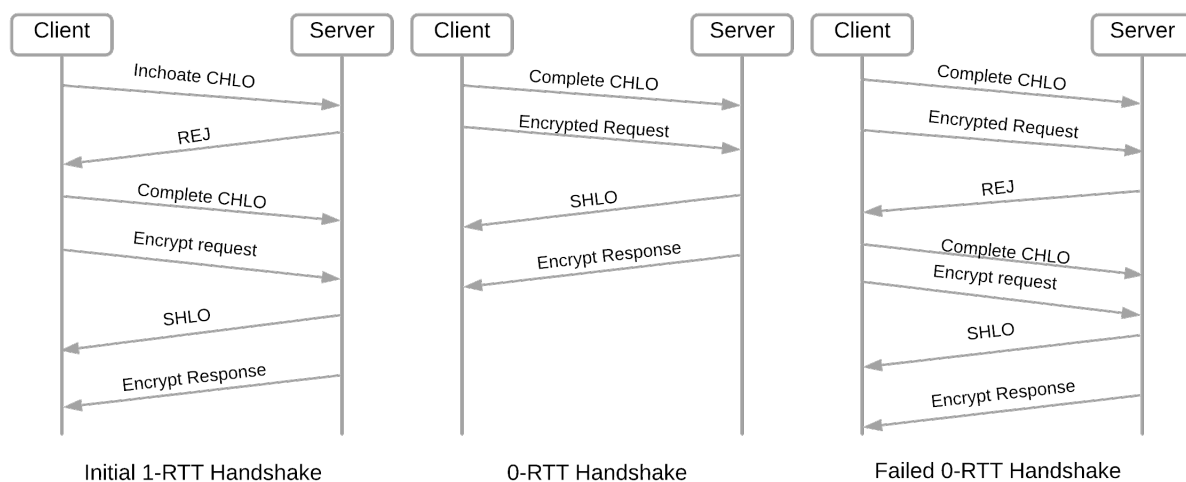


1.2.3 Les principaux concepts autour du protocole QUIC

1.2.3.1 Établissement de connexion

QUIC s'appuie sur une poignée de main (handshake) cryptographique et de transport combinée pour établir une connexion de transport sécurisée. Il utilise une approche de connexion 0-RTT et 1-RTT, permettant d'établir une connexion sécurisée avec une latence minimale [4, 19].

FIGURE 1.3 – QUIC Connection establishment 0-RTT and 1-RTT handshake.[4]



- **0-RTT** : Permet aux clients qui ont précédemment connecté de reprendre une session et d'envoyer des données dès le premier paquet. Cela réduit considérablement le temps nécessaire pour établir une connexion.
- **1-RTT** : Utilisé pour les nouvelles connexions, où une poignée de main initiale est effectuée pour échanger les clés de chiffrement et établir une connexion sécurisée en un seul aller-retour.

La sécurité de l'établissement de connexion est assurée par l'utilisation de TLS, qui garantit que toutes les communications sont chiffrées dès le début de la connexion. QUIC inclut également une négociation de version pour éviter des retards et garantir que le client et le serveur utilisent la même version du protocole. Cela améliore l'efficacité et la sécurité de l'établissement de la connexion.

1.2.3.2 Authentification et Chiffrement

QUIC garantit que presque tous les paquets sont authentifiés et principalement chiffrés, sauf quelques paquets initiaux de la poignée de main et les paquets de réinitialisation[19]. Les parties





non chiffrées de l'en-tête, telles que les indicateurs de paquets, le Connection ID, et le numéro de paquet, sont essentielles pour le routage et le déchiffrement. Toute tentative de manipulation des paquets de poignée de main non chiffrés entraînera l'échec de la connexion, assurant ainsi une sécurité robuste dès l'initialisation.

1.2.3.3 Multiplexage de Flux

Le multiplexage de flux permet à plusieurs flux de données indépendants d'être envoyés simultanément sur une seule connexion QUIC. Chaque flux est identifié de manière unique, ce qui permet une transmission parallèle des données sans interférence entre les différents flux. Cette fonctionnalité élimine le blocage tête de ligne (head-of-line blocking). Contrairement à TCP, où la perte d'un paquet peut bloquer la livraison de tous les paquets suivants, dans QUIC, cette perte de paquets dans un flux n'affecte pas les autres flux, ce qui améliore grandement l'efficacité et la rapidité des transmissions [19].

1.2.3.4 Récupération de Pertes

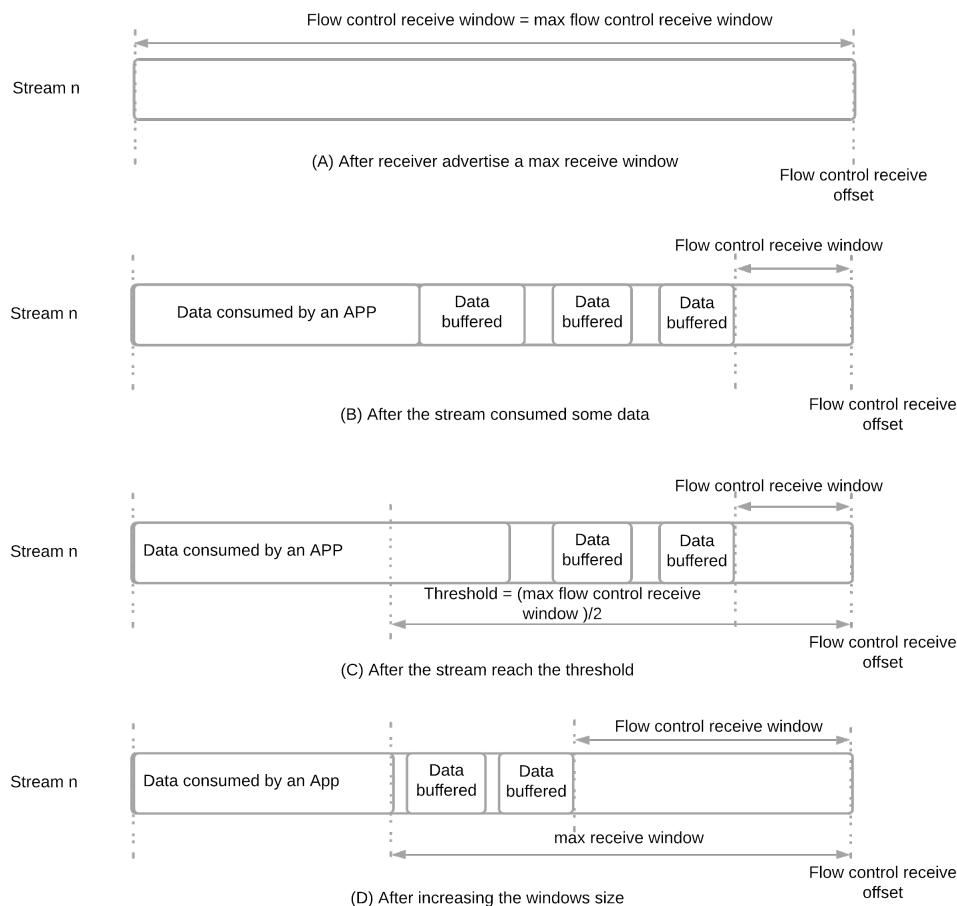
La récupération de pertes dans QUIC est basée sur des algorithmes modernes de détection et de correction des pertes de paquets, conçus pour minimiser l'impact sur la performance de la connexion. QUIC numérote tous les paquets y compris ceux des retransmissions et chaque paquet porte sans distinction un nouveau numéro. Ceci permet d'éviter la nécessité d'un mécanisme distinct des acquittements des retransmissions de ceux des transmissions originales comme c'est le cas dans TCP, il utilise également des décalages de flux dans les trames de flux pour assurer l'ordre de livraison permettant ainsi de séparer ces deux fonctions confondues dans TCP [4, 19].

1.2.3.5 Contrôle de flux

QUIC utilise un contrôle de flux à deux niveaux pour éviter les blocages entre les flux : un au niveau de la connexion globale, et un autre spécifique à chaque flux. Le récepteur limite la quantité de données qu'il est prêt à recevoir en indiquant le décalage maximum d'octets pour chaque flux. Ce mécanisme permet de réguler efficacement le tampon utilisé, évitant ainsi la surcharge du récepteur et permettant une transmission fluide des données[19]. la figure 1.8 illustre ce processus de contrôle de flux.



FIGURE 1.4 – QUIC stream flow control.[4]



1.2.3.6 Contrôle de Congestion

Le contrôle de congestion dans QUIC est conçu pour maximiser le débit tout en s'adaptant aux conditions changeantes du réseau. Il intègre des algorithmes modernes de récupération des pertes, comme le RTO et le F-RTO, et fournit des retours détaillés grâce à l'utilisation de numéros de paquets monotones. QUIC utilise souvent l'algorithme NewReno ou Cubic pour la gestion de la congestion, avec une capacité à maintenir la connexion lors des changements d'adresse IP, grâce à l'ID de connexion, permettant ainsi une migration sans interruption.

1.2.4 Extension de QUIC pour le multipath : MP-QUIC

MP-QUIC (Multipath QUIC) est une extension de QUIC qui permet d'exploiter plusieurs interfaces réseau simultanément. MP-QUIC combine les avantages de QUIC avec les capacités de multipath. Cette extension introduit de nouvelles notions non présentes dans l'implémentation





du QUIC traditionnel qui nous seront utiles pour concevoir notre solution.

MPQUIC présente des avantages conceptuels significatifs. En étant basé sur QUIC, il est plus facile à déployer, car ne dépend pas du système d'exploitation et intègre les informations de contrôle des multiples trajets avec un minimum d'interférence, tout en tirant parti des flux d'applications multiples et des priorités HTTP/2 pour optimiser la transmission des paquets dans des environnements variés [24]. Il existe cependant d'autres protocoles de transport intégrant les fonctionnalités de multipath comme MTCP, SCTP, mais qui nécessitent souvent une prise en charge par le système d'exploitation, ce qui ralentit leur adoption, notamment dans les appareils mobiles.

1.2.4.1 Avantages de MP-QUIC

Bénéficiant déjà des avantages du QUIC traditionnels, le protocole MP-QUIC bénéficie en plus des avantages suivants :

- **Utilisation de multiples chemins réseau** : permet l'utilisation simultanée de plusieurs chemins réseau (par exemple, WiFi et LTE sur un smartphone), augmentant la bande passante globale disponible et améliorant la résilience en cas de défaillance d'une connexion.
- **Amélioration des performances** : Les évaluations montrent qu'il améliore le débit et réduit les temps de téléchargement par rapport à QUIC, TCP et MPTCP [24].
- **Flexibilité et Adaptabilité** : Il peut s'adapter dynamiquement aux conditions changeantes du réseau, basculant entre différentes connexions selon leur disponibilité et leur qualité. Ce qui assure une performance constante même dans des environnements réseau hétérogènes
- **Compatibilité avec les environnements variés** : MP-QUIC est conçu pour fonctionner efficacement dans divers environnements réseau, y compris les réseaux mobiles, les centres de données, et les réseaux domestiques et d'entreprises.

1.2.5 Etude comparative des protocoles de transport QUIC, TCP et UDP

Afin de mettre en perspective les avantages et les inconvénients de QUIC par rapport à TCP et UDP, il est essentiel de comparer ces protocoles sur la base de critères pertinents qui justifient notre choix d'utiliser ce protocole dans notre travail plutôt que les autres. Dans les sections précédentes, nous avons axé notre rédaction de manière à ressortir pour chacun de ces protocoles les aspects utiles à cette section. Pour notre travail qui pour sa globalité portera sur un transfert sécurisé de données ordonnées et nécessitant à la fois une garantie de réception et une grande vitesse d'exécution, nous avons choisi les critères apparents de, la **latence**, la **sécurité**, la **fiabilité**, le **multiplexage de flux**, le **contrôle de flux et de congestion**, la **récupération de**





TABLE 1.1 – Tableau comparatif des protocoles QUIC, TCP et UDP

Critere	QUIC	TCP	UDP
Type de Proto- cole	Orienté connexion sur UDP	Orienté connexion	Sans connexion
Fiabilité	Élevée (via les mécanismes de QUIC)	Élevée	Faible
Établissement de Connexion	Handshake 0-RTT rapide	Handshake en trois temps	Aucun
Multiplexage de Flux	Oui	Non	Non
Contrôle de Flux	Avancé	Standard	Aucun
Contrôle de Congestion	Oui	Oui	Non
Récupération de Perte	Numéros de paquets uniques, ACK explicites	Numéros de séquence, ACK	Aucun
Sécurité	Chiffrement et authentifica- tion intégrés	TCP+TLS externe	Aucun
Flexibilité de Déploiement	Haute, espace utilisateur	Faible, noyau système	Haute, espace uti- lisateur
Résilience	Migration de connexion pos- sible	Non	Non

perles, et la résilience. le tableau 1 . 4 présente cette comparaison etres les protocomes QUIC, TCP et UDP.

1.3 Dimension Énergétique dans les Protocoles de Transport Décentralisés

L'importance de l'efficacité énergétique dans les réseaux de télécommunication est de plus en plus reconnue, en raison des préoccupations croissantes concernant l'empreinte carbone des infrastructures numériques. Cependant, malgré cette prise de conscience, les protocoles réseau traditionnels n'intègrent généralement pas la dimension énergétique dans leur conception.

La consommation énergétique des protocoles actuels est principalement influencée par des facteurs tels que la surcharge de la bande passante, la gestion des connexions, la retransmission des paquets et le contrôle de la congestion au niveau de la couche transport. Ces opérations, bien que nécessaires pour assurer la fiabilité et la performance des transferts de données, peuvent conduire à une utilisation inefficace de l'énergie, en particulier dans les environnements distribués et décentralisés.

Quelques études ont tenté d'estimer l'impact énergétique des protocoles réseau, bien que les chiffres varient selon les méthodologies et les contextes étudiés. D'après Balakrishnan et al.,





2020, Les mécanismes de retransmission et de contrôle de la congestion de TCP, bien qu'essentiels pour la fiabilité, consomment des quantités significatives d'énergie. Une étude a estimé que dans certains environnements, TCP pourrait entraîner jusqu'à 10-15% de surconsommation d'énergie par rapport à des protocoles optimisés pour des tâches spécifiques.

1.3.1 Vers des Protocoles Écoénergétiques

Un des moyens d'améliorer l'efficacité énergétique d'un protocole de transfert de données décentralisé est d'introduire l'usage de sources d'énergies vertes et de prioriser les nœuds fonctionnant aux énergies renouvelables (par exemple, alimentés par le solaire ou l'éolien) pour participer au transfert de données. Cette approche repose sur l'idée que, dans un réseau distribué, tous les nœuds ne sont pas égaux en termes d'empreinte énergétique. Pour améliorer l'efficacité énergétique des transferts dans un mix énergétique (renouvelable + non renouvelable) des mécanismes peuvent être implémentés :

- **Sélection dynamique des nœuds** : Les protocoles pourraient être conçus pour sélectionner les nœuds participant au transfert de données en fonction de leur source d'énergie. Par exemple, un nœud alimenté par l'énergie solaire pourrait être préféré à un autre alimenté par une source d'énergie plus polluante, surtout pendant les périodes de production maximale d'énergie renouvelable.
- **Routing vert** : En fonction de la disponibilité des nœuds écoénergétiques, le protocole pourrait ajuster les routes de transfert de données pour minimiser l'impact énergétique global, même si cela signifie contourner des routes plus directes, mais plus énergivores.

1.4 État de l'art

L'état de l'art dans le domaine des protocoles de transfert de données décentralisés et distribués se concentre sur les solutions qui visent à améliorer la performance, la résilience et l'efficacité énergétique des réseaux modernes. Cette section examine les protocoles actuels, leurs forces et leurs limitations.

1.4.1 Étude des Protocoles Existants

Dans cette section, nous analysons les protocoles de transfert de données en fonction de leur architecture et de leur capacité à répondre aux besoins des environnements décentralisés et distribués.

1.4.2 HTTP

Le HTTP est un protocole de communication destiné aux échanges d'informations sur le Web. Il repose sur un modèle client-serveur où un client envoie des requêtes à un serveur, qui répond





avec les données demandées. Il a été conçu pour faciliter la communication sur le Web. HTTP permet la récupération de documents hypertextes (pages web) et de fichiers multimédias. Son principal objectif est de fournir un mécanisme flexible pour interagir avec les ressources en ligne.

Principe de Fonctionnement Le HTTP fonctionne sur une connexion TCP et utilise une architecture de requête-réponse. Le client, généralement un navigateur web envoie des requêtes HTTP (par exemple, GET, POST) et le serveur répond avec des ressources, comme des pages HTML ou des fichiers JSON. Chaque transaction HTTP est stateless (sans état), c'est-à-dire que chaque requête est indépendante de la précédente.

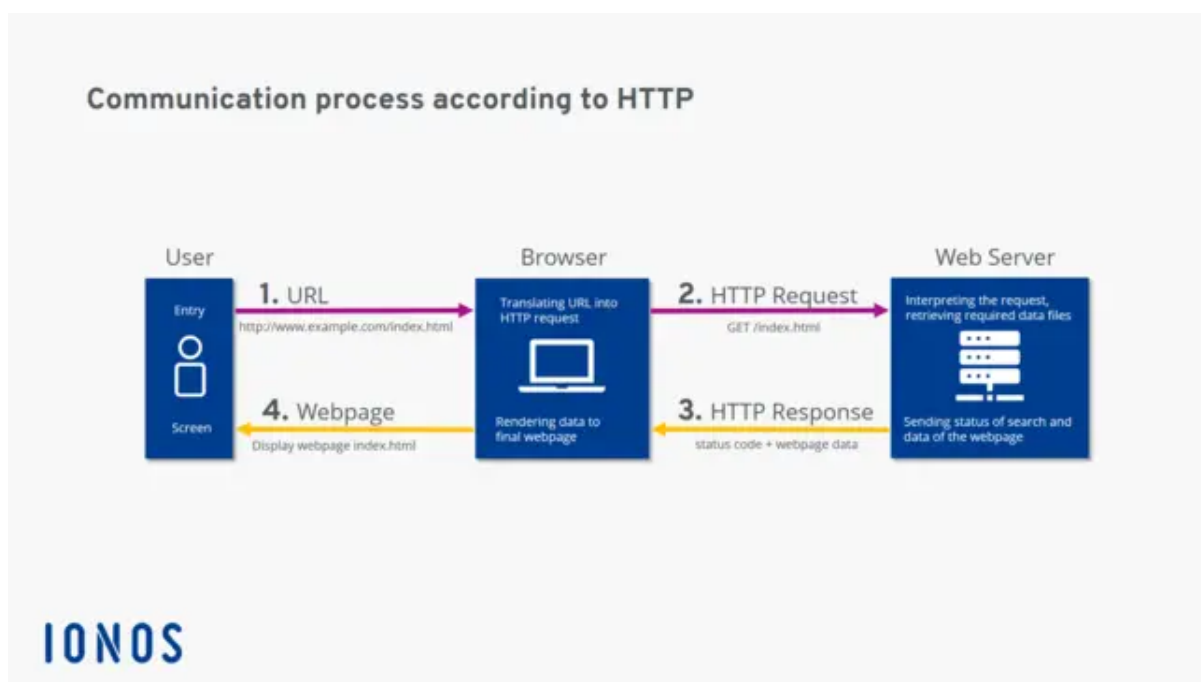


FIGURE 1.5 – Processus de communication HTTP.[18]

1.4.3 FTP

FTP[14] est un protocole standard utilisé pour transférer des fichiers entre un client et un serveur sur un réseau TCP/IP. Il permet à un utilisateur d'uploader ou de télécharger des fichiers. FTP a été conçu pour permettre un transfert fiable de fichiers volumineux entre des systèmes distants, tout en offrant des fonctionnalités basiques d'authentification.

Principe de Fonctionnement FTP utilise deux canaux de communication[23] : un canal de commande (port 21) pour échanger des commandes et un canal de données (port 20 ou dynamique) pour le transfert des fichiers. Il fonctionne en mode actif (où le client accepte une connexion depuis le serveur pour les données) ou passif (où le client initie la connexion de données).



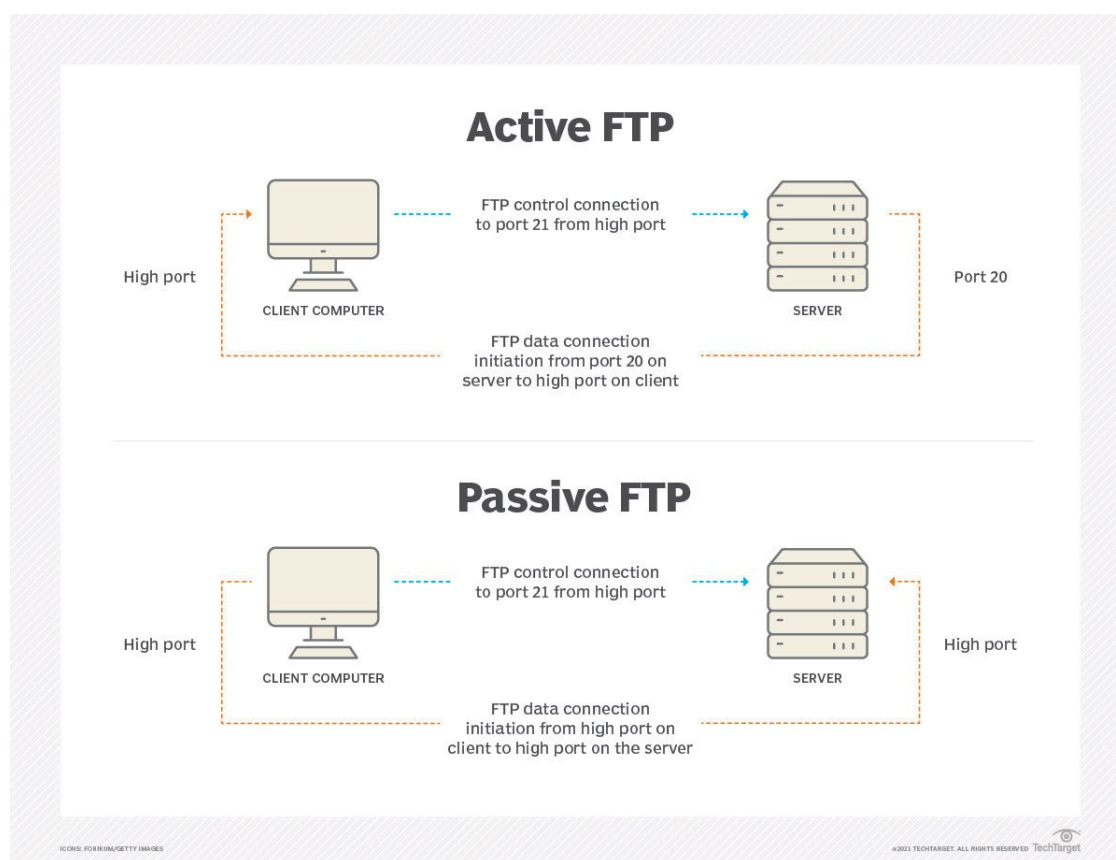


FIGURE 1.6 – Fonctionnement modes actif et passif du protocole FTP.[23]

1.4.4 SFTP

SFTP est une version sécurisée de FTP fonctionnant au-dessus du protocole SSH (Secure Shell). Il assure le transfert de fichiers avec des mécanismes de sécurité avancés. SFTP a été conçu pour combiner les fonctionnalités de FTP avec la sécurité de SSH, garantissant ainsi la confidentialité et l'intégrité des fichiers échangés.

Principe de Fonctionnement SFTP utilise une seule connexion sécurisée sur le port 22, contrairement à FTP qui en utilise deux. Toutes les données, y compris les commandes et les fichiers, sont chiffrées via SSH. Les utilisateurs doivent s'authentifier via SSH pour accéder aux fichiers distants, et toutes les actions de gestion de fichiers sont effectuées de manière sécurisée. L'architecture du protocole repose sur les éléments clés suivants :

- **Client SFTP** : Le logiciel utilisé pour établir une connexion SSH sécurisée et gérer les fichiers.
- **Serveur SFTP** : L'hôte sécurisé qui accepte les connexions SFTP et assure le transfert sécurisé des fichiers.





- **SSH** : Le protocole sous-jacent qui fournit le chiffrement et l'authentification des connexions.

1.4.5 GridFTP

GridFTP[22] est une extension du protocole FTP, conçue pour le transfert de fichiers à grande échelle dans des environnements distribués comme les grilles de calcul et les centres de données. Il améliore FTP en y ajoutant des fonctionnalités adaptées aux réseaux à large bande passante et à haute latence[5, 10], permettant ainsi une meilleure gestion des transferts massifs de données.

Principe de fonctionnement GridFTP utilise plusieurs connexions parallèles pour optimiser la bande passante disponible lors du transfert de données volumineuses. Il inclut des mécanismes tels que la reprise automatique des transferts après interruption et la gestion des connexions multiples. Il est largement utilisé dans les grilles de calcul scientifique, où le transfert rapide et fiable de données massives est critique[5].

Caractéristiques principales

- **Parallélisme** : Utilisation de multiples connexions pour augmenter le débit de transfert.
- **Reprise automatique** : Les transferts interrompus peuvent reprendre là où ils ont été arrêtés.
- **Optimisation pour haute latence** : Conçu pour fonctionner efficacement sur des réseaux à haute latence.
- **Sécurité** : Supporte les extensions de sécurité comme GSI (Grid Security Infrastructure) pour l'authentification et le chiffrement.

Cas d'usage typique GridFTP est principalement utilisé dans les environnements de calcul intensif, les centres de données, et les réseaux distribués pour le transfert de fichiers volumineux dans des infrastructures de grilles, où la fiabilité et l'efficacité sont essentielles.

1.4.6 BitTorrent

BitTorrent est un protocole peer-to-peer (P2P) conçu pour le partage de fichiers de manière distribuée[3, 11]. Contrairement aux approches centralisées, BitTorrent permet à chaque participant (appelé "pair") de télécharger et de partager simultanément des parties d'un fichier, ce qui rend le protocole extrêmement efficace pour la distribution de fichiers volumineux à un grand nombre d'utilisateurs.





Principe de fonctionnement[11] Le fichier à partager est divisé en petits morceaux. Chaque pair télécharge les morceaux disponibles auprès des autres pairs et, en parallèle, partage les morceaux qu'il a déjà téléchargés. Cela réduit la dépendance à un seul serveur central et améliore la robustesse et la vitesse des transferts à mesure que le nombre de pairs augmente.

Caractéristiques principales

- **Décentralisation** : Aucun serveur centralisé n'est requis, la distribution des fichiers est répartie entre les pairs.
- **Partage simultané** : Les utilisateurs téléchargent et partagent des morceaux de fichier en même temps.
- **Évolutivité** : Plus il y a de pairs dans le réseau, plus le transfert est rapide.
- **Tolérance aux pannes** : La perte de quelques pairs n'affecte pas la capacité à télécharger le fichier.

Cas d'usage typique BitTorrent est couramment utilisé pour la distribution de fichiers volumineux à grande échelle, comme les logiciels open source, les distributions Linux, ou encore des médias volumineux. Il est particulièrement adapté aux environnements où le téléchargement par un grand nombre de personnes est nécessaire sans surcharger un serveur central.

1.4.7 FDT

FDT (Fast Data Transfer) est un protocole optimisé pour le transfert de fichiers à haut débit sur des réseaux longue distance. Il est principalement utilisé dans les contextes où les volumes de données sont massifs, comme les grandes expériences scientifiques [20]. FDT exploite pleinement la bande passante disponible sur les réseaux à longue distance, en utilisant des connexions multiples pour maximiser le débit.

Principe de fonctionnement FDT utilise plusieurs canaux TCP parallèles pour assurer un transfert rapide des fichiers volumineux. Il est capable de segmenter un fichier en flux multiples et de gérer la synchronisation de ces flux sur des réseaux à haute latence. Il optimise également l'utilisation de la bande passante, ce qui permet d'éviter les goulots d'étranglement sur les réseaux longue distance.[22]

Caractéristiques principales

- **Parallélisme des flux** : Utilisation de plusieurs flux TCP simultanés pour maximiser l'utilisation de la bande passante.
- **Optimisation pour longue distance** : Conçu pour des transferts à très haute vitesse sur des réseaux à grande distance géographique.





- **Reprise automatique** : Capacité à reprendre les transferts interrompus.
- **Gestion automatique des connexions** : Ajustement dynamique du nombre de connexions pour optimiser le débit.

Cas d'usage typique FDT est souvent utilisé dans les grandes infrastructures scientifiques et académiques, par exemple pour le transfert de données entre centres de recherche géographiquement éloignés ou pour les expériences dans le domaine de la physique des particules, où des volumes énormes de données doivent être transférés rapidement.

1.4.8 MDTMFTP

MDTMFTP[22, 25] est une extension de FTP qui introduit des optimisations pour les transferts de données à travers des réseaux distribués. Il améliore les performances de FTP en exploitant le parallélisme et la gestion de flux multiples, tout en conservant la compatibilité avec les fonctionnalités de base de FTP, comme la gestion de fichiers et les contrôles d'accès.

Principe de fonctionnement MDTMFTP conserve le mécanisme de base de FTP mais optimise les transferts en gérant plusieurs connexions simultanées pour le transfert des fichiers. Il utilise des méthodes de compression et des ajustements dynamiques pour réduire les temps de transfert et gérer plus efficacement la bande passante disponible.

Caractéristiques principales

- **Parallélisme des connexions** : Utilisation de connexions multiples pour améliorer la vitesse de transfert.
- **Compression des données** : Réduction de la taille des fichiers pour accélérer les transferts.
- **Reprise automatique** : Support de la reprise des transferts en cas d'interruption.
- **Compatibilité FTP** : Conserve toutes les fonctionnalités de base de FTP (gestion des fichiers, authentification).

Cas d'usage typique MDTMFTP est particulièrement adapté aux environnements où il est nécessaire de transférer des fichiers volumineux de manière fiable et rapide sur des réseaux distribués. Il est couramment utilisé dans les centres de données, les infrastructures cloud, et les réseaux de calcul intensif.

1.4.9 Limites des Protocoles Existants

L'analyse des protocoles existants met en lumière plusieurs limitations, particulièrement dans les environnements décentralisés :





- **Scalabilité** : Les protocoles basés sur une architecture client-serveur, comme FTP, HTTP, et même SFTP, rencontrent des limites de scalabilité, en raison de leur dépendance à des serveurs centraux. Bien que des extensions comme GridFTP et MDTMFTP introduisent des améliorations, elles restent contraintes par leur architecture centralisée.
- **Résilience** : Les protocoles P2P comme BitTorrent offrent une meilleure résilience grâce à leur nature décentralisée, mais cette architecture expose également à des risques accrus de sécurité, tels que les attaques malveillantes.
- **Efficacité énergétique** : La consommation énergétique n'est pas un critère pris en compte dans la conception de la plupart des protocoles actuels, même ceux optimisés pour des transferts rapides comme FDT. Dans un contexte où l'empreinte carbone des infrastructures numériques devient un enjeu majeur, cette lacune doit être adressée.

1.4.10 Impact du Protocole QUIC

Le protocole de transport moderne QUIC, vise à surmonter certaines des limitations des protocoles existants, en offrant des améliorations significatives en termes de performance, sécurité et résilience. En particulier, les extensions comme MP-QUIC introduisent des capacités de multi-chemin, améliorant la performance dans des environnements distribués tout en offrant une meilleure gestion de la congestion et du flux de données. QUIC est ainsi un bon support pour bâtir un protocole de transfert de données, car il est plus performant pour le transport que TCP et UDP mais à l'heure actuelle, nous n'avons pas connaissance des travaux d'amélioration sur les protocoles existants pour prendre en compte QUIC à l'exception des travaux sur HTTP3[4, 19] qui sont en cours.

1.4.11 Étude Comparative des protocoles de transfert de données existant



TABLE 1.2 – Comparaison des protocoles de transfert de données existant à QuicPlex

Caractéristiques	GridFTP	BitTorrent	FDT	MDTMFTP	HTTP/2	FTP
Type de protocole	FTP optimisé pour les grilles	P2P (Pair-à-Pair)	Optimisé pour transfert à haut débit	FTP amélioré pour réseaux larges	Protocole d'application (HTTP)	Protocole d'application
Multiplexage	Oui, via connexions multiples	Oui, via les pairs	Oui, plusieurs flux TCP	Oui, plusieurs flux FTP	Oui, via une seule connexion TCP	Non
Transfert P2P	Non	Oui	Non	Non	Non	Non
Bande passante	Optimisée avec plusieurs connexions	Partagée entre pairs	Maximisée via flux multiples	Maximisée via flux multiples	Optimisée par la gestion des flux	Fixe, sans optimisation
Tolérance aux pannes	Oui, reprise automatique	Oui, via les pairs	Oui, reprise automatique	Oui, reprise automatique	Non	Non
Sélection des relais	Non applicable	Dynamique (pairs aléatoires)	Non applicable	Non applicable	Non applicable	Non applicable
Gestion de l'énergie verte	Non	Non	Non	Non	Non	Non
Priori-sation des flux	Non	Non	Oui	Non	Oui	Non
Sécurité	Sécurité optionnelle	Variable (dépend des clients)	Sécurité optionnelle	Sécurité optionnelle	Chiffrement (TLS)	Sécurité de base (authentification)
Latence	Moyenne, selon les conditions	Variable (dépend du réseau)	Faible (optimisé pour réseaux à longue distance)	Faible (optimisé pour grandes distances)	Faible (optimisé pour le web)	Haute
Usage typique	Transfert massif dans les grilles	Partage de fichiers volumineux	Transferts massifs scientifiques	Transfert de fichiers distribués	Communication client-serveur (web)	Transfert de fichiers simple



1.5 Bilan du chapitre et Positionnement

1.5.1 Bilan

Ce chapitre a examiné les principaux protocoles de transport et de transfert de données, avec un focus sur QUIC. Nous avons montré comment QUIC surmonte les limitations de TCP et UDP grâce à ses mécanismes avancés de contrôle de flux, de congestion, et de récupération des pertes. En comparant QUIC avec d'autres protocoles, nous avons mis en évidence ses avantages en termes de performance et de sécurité. De plus, notre analyse des protocoles comme HTTP, FTP, SFTP, BitTorrent, FDT, et MDTMFTP a révélé un manque de prise en compte des considérations énergétiques et la non utilisation du protocole QUIC comme protocole de transport[5, 14, 25] soulignant un besoin crucial pour des recherches futures dans ce domaine.

1.5.2 Positionnement

Notre travail vise à combler les lacunes identifiées, notamment l'absence de mécanismes d'optimisation énergétique et l'intégration insuffisante de QUIC dans les environnements décentralisés. Nous proposons de développer un nouveau protocole qui exploitera les avantages de QUIC tout en intégrant des mécanismes de décentralisation et d'efficacité énergétique. Cette approche répondra aux besoins croissants en matière de transfert de données tout en minimisant la consommation d'énergie, un aspect critique rarement abordé par les protocoles actuels.



CONCEPTION DU PROTOCOLE DATASTREAMX

Dans ce chapitre, nous décrivons la conception du **protocole DataStreamX**, en nous basant sur les problématiques identifiées au chapitre précédent. Nous détaillerons l'**architecture décentralisée** et **modulaire** du protocole, en montrant comment elle permet de répartir les flux de données entre plusieurs relais de manière efficace. Enfin, nous présenterons les **algorithmes adaptatifs** développés pour optimiser les connexions multiplexées, gérer les fluctuations de la bande passante et synchroniser le transfert décentralisé des données.



2.1 Rappel du problème et des objectifs poursuivis

Le protocole *DataStreamX* est conçu pour résoudre deux principaux problèmes associés aux protocoles de transfert de données classiques : l'efficacité énergétique et la réduction du temps de transfert. Dans un monde où la quantité de données échangées ne cesse de croître, la nécessité d'optimiser ces deux aspects devient primordiale, tant pour les performances des réseaux que pour leur impact environnemental.

Les approches centralisées traditionnelles rencontrent des limitations face à ces enjeux, notamment en raison d'une consommation d'énergie élevée et d'une inefficacité lors des transferts de grandes quantités de données sur divers réseaux. L'objectif du protocole *DataStreamX* est de combiner une gestion écoénergétique avec une approche décentralisée, tout en s'appuyant sur la flexibilité et la robustesse du *protocole QUIC*.

2.1.1 Objectif général

Le protocole *DataStreamX* a pour objectif général d'optimiser simultanément le *temps de transfert* des données entre un serveur et un client et la *performance énergétique* du réseau en intégrant des relais. Ces relais, similaires aux pairs du protocole BitTorrent, sont des serveurs possédant une copie partielle ou totale des données et peuvent intervenir dans le processus de transfert. La coordination entre les relais est assurée par le serveur central, qui optimise la répartition des segments de données en fonction de critères de performance tels que la *bande passante* et la *disponibilité de l'énergie verte*.

2.1.2 Modélisation mathématique du problème d'optimisation

2.1.2.1 Variables et paramètres du modèle

Le protocole *DataStreamX* peut être formalisé à l'aide des variables et paramètres suivants :

- S : Taille totale des données à transférer (en octets).
- F : Taille de la fenêtre de réception (en octets).
- B_i : Bande passante disponible pour chaque relais i (en octets par seconde).
- T_i : Temps pris par le relais i pour transférer son segment de données (en secondes).
- P_i : Puissance consommée par le relais i pour transférer son segment donnée qui lui est alloué (en joules).
- E_i : Indicateur de la disponibilité de l'énergie verte pour le relais i (1 si énergie verte disponible, 0 sinon).
- α : Pondération de l'importance du temps de transfert.
- β : Pondération de l'importance de la consommation énergétique.
- N : Nombre total de relais disponibles.





- d_i : Portion de la donnée totale assignée au relais i (en octets).
- L_i : Latence du relais i (en secondes).

2.1.2.2 Objectifs de l'optimisation

Nous cherchons à atteindre deux objectifs concurrents : la *minimisation du temps de transfert* et la *réduction de la consommation énergétique*. Ces deux objectifs sont définis mathématiquement comme suit :

2.1.2.3 Temps total de transfert T_{total}

$$T_{total} = \max_{i \in \{1, 2, \dots, N\}} \left(\frac{d_i}{B_i} + L_i \right)$$

Le temps de transfert total correspond au temps pris par le relais le plus lent à compléter son envoi de données au client.

2.1.2.4 Consommation énergétique totale P_{total}

$$P_{total} = \sum_{i=1}^N \left(P_i \cdot \frac{d_i}{B_i} \right)$$

La consommation énergétique totale correspond à la somme des énergies dépensées par chaque relais pour transférer ses données, proportionnellement au temps qu'il met à compléter son segment.

Formulation de la fonction objectif Nous cherchons à *minimiser simultanément* ces deux objectifs. Cela peut être formulé comme un problème d'optimisation multiobjectifs pondérés :

$$\min(\alpha \cdot T_{total} + \beta \cdot P_{total})$$

où α et β sont des pondérations représentant l'importance relative accordée au temps de transfert et à la consommation énergétique, respectivement. Ces poids peuvent être ajustés en fonction des priorités du système (i.e., priorité donnée à la rapidité ou à l'efficacité énergétique).

2.1.3 Contraintes du problème

Le protocole DataStreamX est soumis à plusieurs contraintes qui doivent être intégrées dans l'optimisation :





2.1.3.1 Contraintes de bande passante

Chaque relais ne peut pas dépasser sa capacité de bande passante.

$$\frac{d_i}{B_i} \leq T_{max} \quad \forall i \in \{1, 2, \dots, N\}$$

où T_{max} est le temps maximal toléré pour le transfert de chaque portion de données.

2.1.3.2 Disponibilité de l'énergie verte

Le système doit privilégier les relais alimentés par des sources d'énergie verte, selon une fraction γ de la taille totale des données.

$$\sum_{i=1}^N E_i \cdot d_i \geq \gamma \cdot S$$

où $\gamma \in [0, 1]$ représente la proportion minimale de données transférée via des relais alimentés en énergie verte.

2.1.3.3 Intégrité des données

Chaque segment de données doit être vérifié et confirmé avant la fin de la transmission.

$$\sum_{i=1}^N \mathbb{1}(\text{segment vérifié pour } d_i) = 1 \quad \forall i$$

2.1.4 Complexité du problème

Le problème posé par DataStreamX consiste à minimiser simultanément le **temps de transfert** et la **consommation énergétique** en répartissant les segments de données entre des relais, tout en respectant des contraintes de bande passante, de latence et de disponibilité énergétique. Ce problème entre dans la catégorie des **problèmes d'optimisation combinatoire**, où de multiples facteurs interdépendants doivent être optimisés simultanément.

Deux éléments contribuent à la complexité du problème :

1. **Sélection optimale des relais** : Choisir les relais de manière optimale pour un transfert rapide et économe en énergie, un problème qui ressemble à l'allocation de ressources ou au job scheduling, qui sont des problèmes NP-difficiles.
2. **Répartition des segments** : Répartir les segments de données entre les relais de manière à minimiser les temps de transfert est similaire au bin packing problem, lui-même NP-difficile.

Le problème de DataStreamX peut être réduit à des problèmes connus comme le **job scheduling** ou le **bin packing**, qui sont NP-complets. Ainsi, notre problème est au moins aussi difficile que ces derniers. Les détails de conception du protocole et les algorithmes proposés dans les sections suivantes tendent à apporter une solution satisfaisante et calculable en temps raisonnable à ce problème.





2.2 Modélisation UML du protocole DataStreamX

Le protocole DataStreamX repose sur trois principaux composants : un serveur, un client, et plusieurs relais. Ces derniers jouent un rôle crucial dans le partage de la charge de transfert de données, en participant activement à la transmission selon leur capacité et leur disponibilité en énergie verte.

Le serveur joue un rôle central dans la coordination des échanges, en sélectionnant les relais les plus efficaces pour chaque session. Le client, quant à lui, initie les requêtes de données et reconstitue les fichiers à partir des segments reçus de différents relais. Chaque transfert est suivi d'une série de confirmations et de rapports afin d'assurer la fiabilité des échanges.

2.2.1 Spécification des exigences

2.2.1.1 Besoins fonctionnels

- **Transmission de données par segments** : Le système doit diviser les fichiers en segments et transférer ces segments via plusieurs relais.
 - Chaque segment doit être assigné à un relais en fonction de sa bande passante et de sa latence.
 - Le client doit être capable de reconstituer le fichier original à partir des segments reçus.
- **Sélection dynamique des relais** : Le serveur doit sélectionner dynamiquement les relais pour optimiser le temps de transfert et la consommation énergétique.
 - Critères de sélection des relais : bande passante, latence, disponibilité de l'énergie verte, historique de performance.
 - Ré-attribution automatique des segments en cas de défaillance d'un relais.
- **Transmission fiable et récupération d'erreurs** : Chaque segment de données doit être transmis avec un haut degré de fiabilité.
 - Mécanisme de *checksum* pour vérifier l'intégrité des segments reçus.
 - Retransmission automatique en cas de perte ou de corruption d'un segment.
- **Coordination du transfert par le serveur** : Le serveur doit coordonner la transmission des segments par les relais au client, en suivant un modèle de fenêtres de réception.
 - Calcul du nombre de fenêtres de réception pour un transfert donné.
 - Synchronisation du transfert de chaque fenêtre.

2.2.1.2 Besoins non fonctionnels

- **Optimisation du temps de transfert** : Le système doit minimiser le temps total de transfert en maximisant le parallélisme et en réduisant les latences.





- Maximiser l'utilisation de la bande passante disponible.
- Répartir les segments de manière optimale pour éviter les goulots d'étranglement.
- **Efficacité énergétique** : Le protocole doit prioriser l'utilisation des relais alimentés par de l'énergie verte.
 - Sélectionner les relais disposant de sources d'énergie renouvelables.
 - Minimiser la consommation énergétique globale pendant le transfert des données.
- **Adaptabilité et évolutivité** : Le système doit pouvoir s'adapter aux changements dans les conditions du réseau (ajout ou suppression de relais) sans interrompre la session de transfert.
- **Tolérance aux pannes** : Le protocole doit pouvoir récupérer automatiquement des pertes de données ou des défaillances de relais.
 - Mise en place de mécanismes de redondance pour compenser les segments perdus.
 - Utilisation des relais disponibles pour retransmettre les segments défectueux.

2.2.2 Analyse

Nous pouvons représenter les exigences du protocole sous forme de scénario modélisé avec un diagramme d'activité. (voir figures 2.1)

La figure 2.1 décrit les processus d'établissement d'une session et de communication entre le client (respectivement le relais) et le serveur. la partie cliente ouvre une session que la partie serveur peut accepter ou refuser. il peut également se produire un événement **TimeOut** qui au bout d'un certain temps sans réponse du serveur va annuler la demande d'ouverture de session et terminer le processus. Et si le serveur accepte la demande de session du client, une connexion est alors établie entre les deux parties les permettant de commencer l'échange de données. Ils peuvent également fermer la session à la fin de leur échange.

En plus de l'ouverture de session, nous explicitons le fonctionnement général de DataStreamX afin de répondre aux exigences fonctionnelles citées plus haut. Nous pouvons constater ici que chaque relais établit initialement une session avec le serveur, cette session permet au serveur de disposer d'une table de relais parmi lesquelles il fera la sélection lors des transferts, ceci lui permet également de savoir si un relais est encore actif ou pas.

Le point de départ du protocole DataStreamX est la partie cliente. En effet, Le client après ouverture de sa session avec le serveur effectue une requête de données à ce dernier et attend sa réponse. Lorsque le serveur reçoit une requête de données d'un client, il procède à la sélection des relais pouvant desservir le client, puis il renvoie les méta données du fichier au client et en parallèle, il construit les fenêtres coulissantes sur les données, répartit les segments entre les relais et procède à la coordination des relais. Les relais restent en attente de requêtes du serveur, jusqu'à ce qu'ils reçoivent des requêtes de clonage/création de session, de fin transfert et des requêtes de transfert de segments. Une requête clonage/création de session, permet d'établir





de manière transparente ou non une session avec le client. La transparence vient du fait que le relai peut s'identifier comme un chemin vers le serveur. Lorsqu'un relai reçoit une requête de transfert de segment de données, il récupère les données réelles contenues entre les deux extrémités incluses de son segment et les envoie au client, puis il renvoie un acquittement au serveur concernant ce segment lorsque toutes les données sont envoyées. À la réception d'une requête de fin de transfert, il s'arrête et revient au point initial ou ferme sa session s'il en reçoit l'ordre.

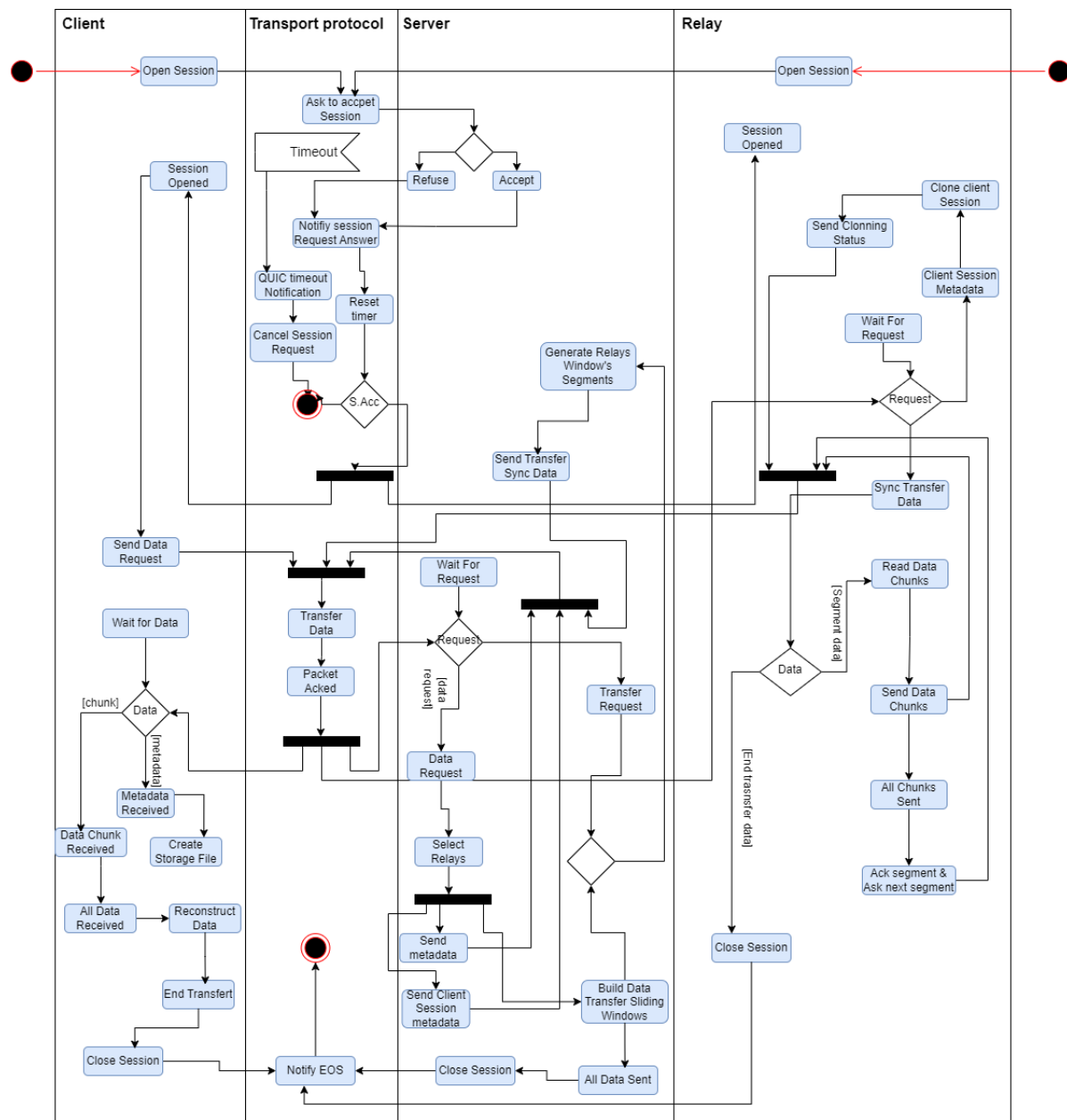


FIGURE 2.1 – Diagramme d'activité global du protocole DataStreamX





Lorsque le client reçoit les méta données, il crée un fichier à partir de celles-ci pour écrire les données réelles reçues. À la réception d'une portion de données réelle, le client va vérifier l'intégrité de la donnée grâce au checksum du paquet reçu et l'écrire directement dans le fichier s'il s'agit de la prochaine donnée attendue ou la stocker dans un fichier temporaire pour une écriture ultérieure, il reconstitue les données initiales de cette façon durant la réception. il enverra des requêtes de retransmission de paquets si de trop grands espacements entre les données reçues surviennent pour ne pas les oublier. Lorsque les données reçues sont correctement reconstituées, il envoie une requête de fin de transfert et s'arrête. Cette requête de transfert signale la terminaison du processus de transfert. cette analyse du protocole nous permet de ressortir les diagrammes d'état transition de ses composants.

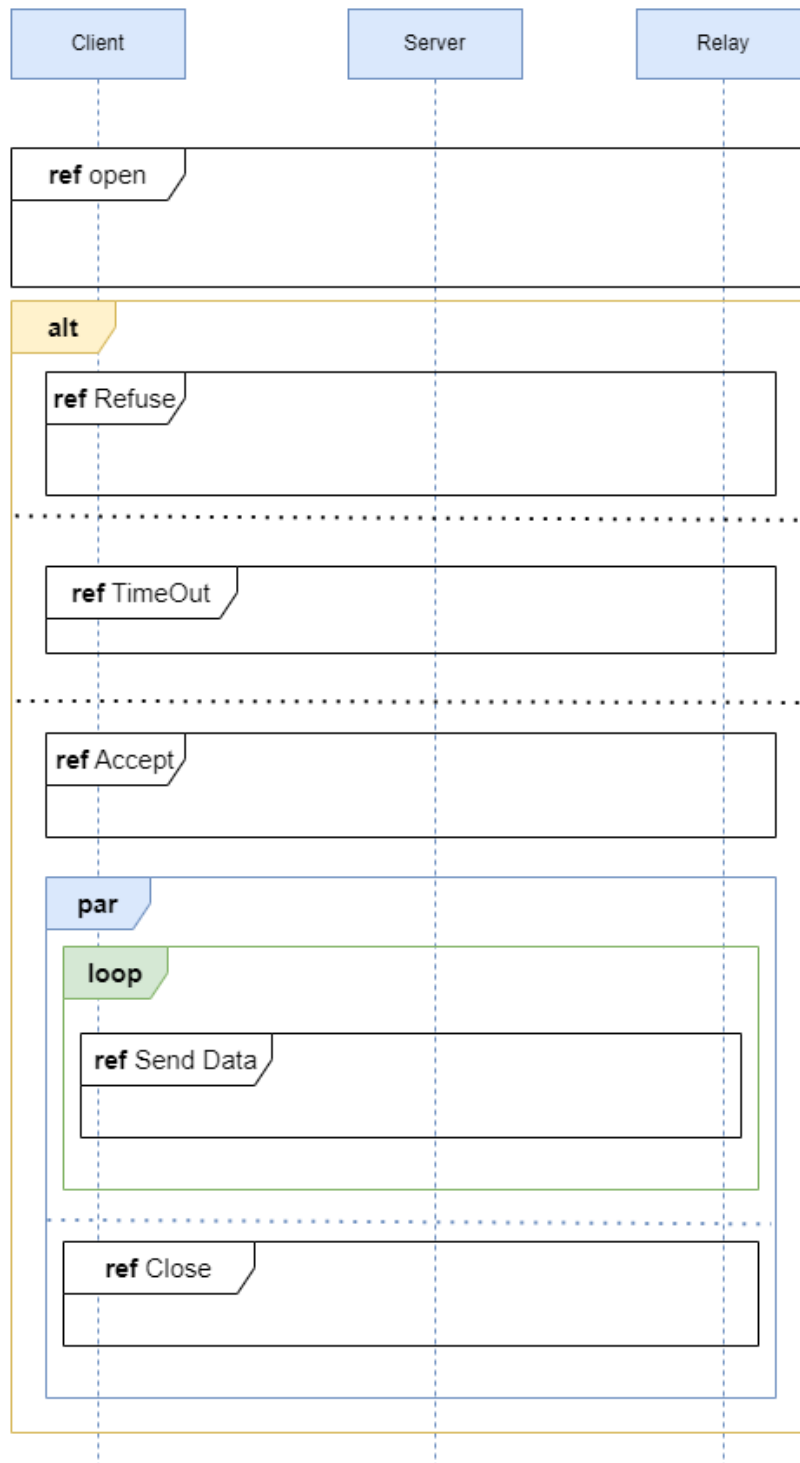
2.2.2.1 Diagramme de Séquence

Afin de créer les machines d'état représentant le fonctionnement du protocole DataStreamX, il serait très utile de disposer de la liste des messages échangés entre les composantes du protocole ; de plus, nous devons disposer de cette liste dans un ordre chronologique et organisée par scénario. Pour obtenir cette liste de messages, nous allons élaborer un diagramme de séquence basé sur le diagramme d'activités de la figure 2.1. Le problème est qu'un seul diagramme de séquence représentant toutes les activités et toutes les combinaisons possibles pourrait être très long et difficile à analyser. Pour éviter un diagramme de séquence très long, nous donnons la structure générale du diagramme de séquence (figure 2.2)



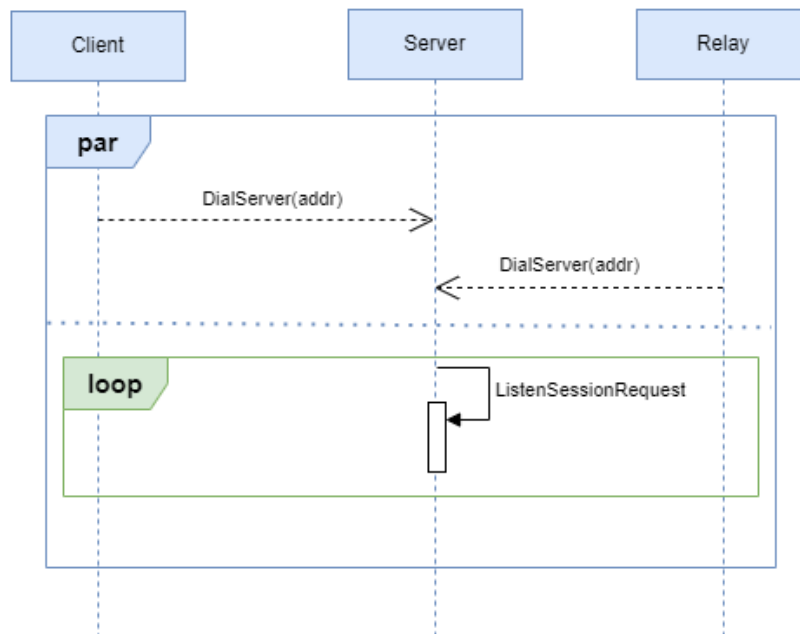


FIGURE 2.2 – Diagramme de séquence DataStreamX



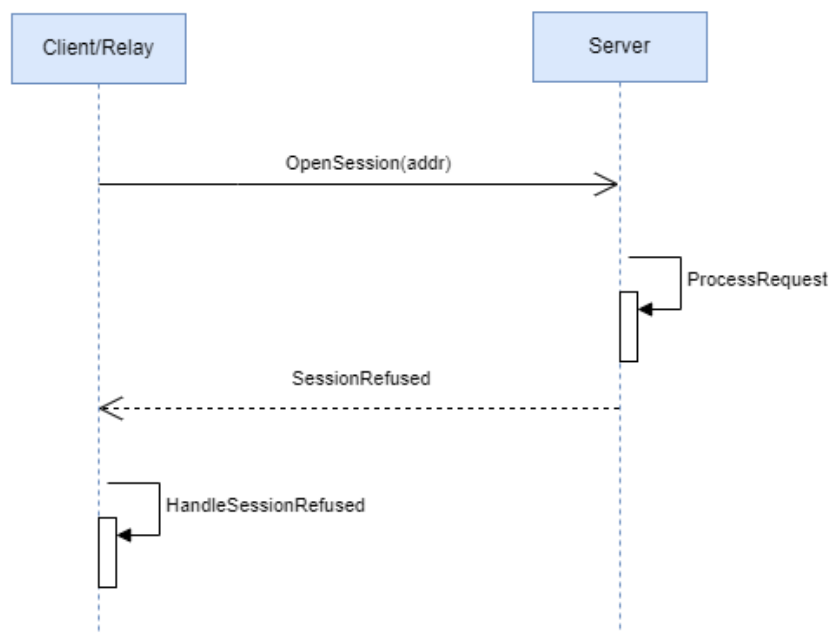
La figure 2.2 montre la segmentation que nous allons appliquer au diagramme de séquence du fonctionnement global du protocole DataStreamX puis dans la suite, nous modélisons parties importantes de ce diagramme.

FIGURE 2.3 – Diagramme de séquence Ouverture de session



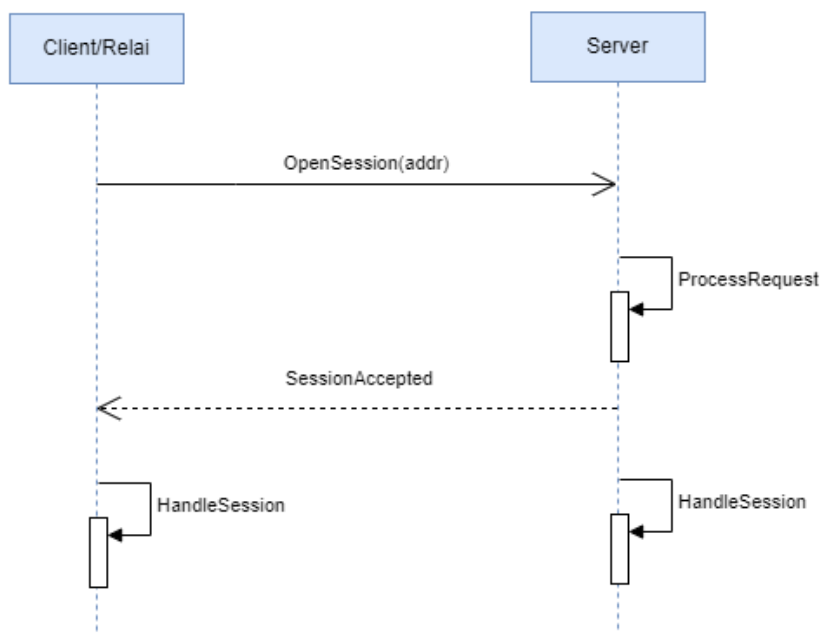
Dans le diagramme de la figure 2.3 le serveur écoute les requêtes venant à la fois des clients et des relais, les clients et les relais sont tous d'un point de vue architectural des clients pour le serveur. lorsque le serveur reçoit une requête d'ouverture de session, il l'accepte ou la refuse, le refus de l'ouverture de session peut être dû à un problème réseau. Comme l'indique la figure 2.4 le refus de la session est notifié au client par un message, il est important de rappeler que cet aspect est entièrement géré par le protocole de transport, soit le protocole QUIC dans notre cas.

FIGURE 2.4 – Diagramme de séquence Refus de session



L'acceptation de la session entre le client et le serveur établit la connexion au niveau de la couche transport entre ces deux composants. il en est de même pour le serveur. Comme dit plus haut, lors du clonage de session au niveau du relais, on bénéficie de la connexion déjà établie entre le serveur et le client et on ajoute le relais comme chemin vers le serveur et on le configure pour la concordance des clés cryptographiques. le processus est le même que ce qui précède s'il s'agit d'une connexion directe entre le client et le relais. La figure 2.5 montre que lorsque la session est créée, chacun des composants peut soit passer à l'exécution proprement dite du protocole, soit fermer la session.

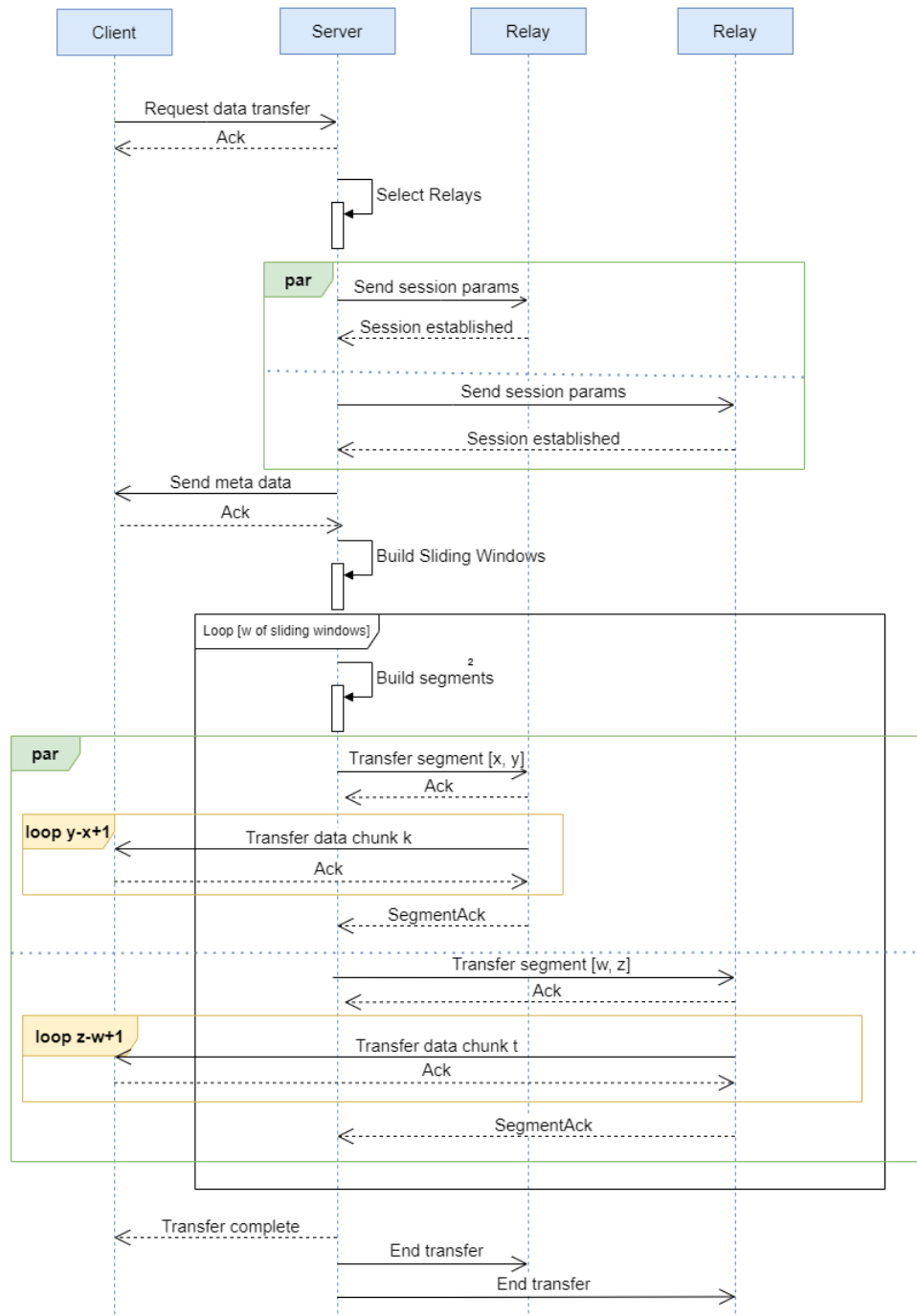
FIGURE 2.5 – *Diagramme de séquence Acceptation de session*



Le diagramme de séquence d'envoi de données de la figure 2.5 illustre le fonctionnement de DataStreamX conformément aux exigences fonctionnelles et l'analyse du protocole faite à la section 2.3.2. On y retrouve la parallélisation des communications entre le serveur et les relais, mais aussi le transfert parallèle et distribué des envois de données des relais vers le client. Ce diagramme fait également apparaître les acquittements qui sont ici pour la plupart résultants de QUIC.



FIGURE 2.6 – Diagramme de séquence envoi de données



Ces différents diagrammes de séquences sont nécessaires pour ressortir la liste des signaux utilisés par le DataStreamX pour organiser les échanges entre ses composants. Ils permettent également de voir les responsabilités de chaque composant dans le transfert des données, notam-



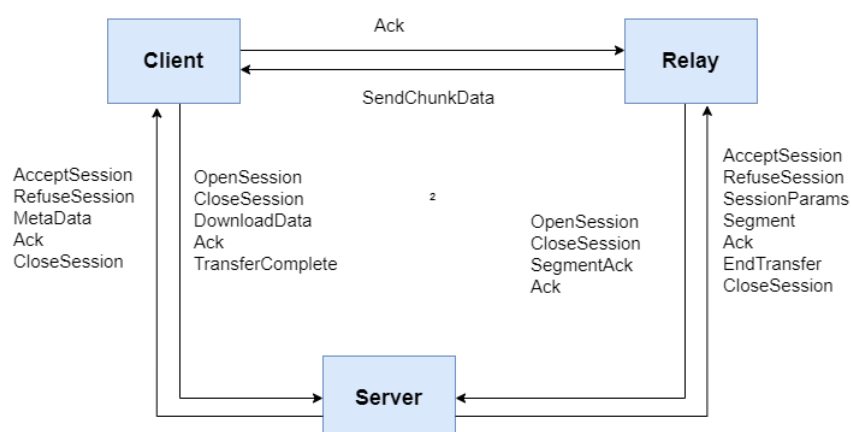


ment celle du serveur dans la coordination du transfert.

2.2.2.2 Liste des signaux

Sur la base des spécifications, des exigences et des diagrammes de séquence que nous avons précédemment décrits dans la section 2.3.2.1, il nous a été possible de définir un ensemble de messages échangés entre le serveur, le client et les relais. Ces messages sont représentés dans la figure 2.7.

FIGURE 2.7 – Liste des signaux échangée dans DataStreamX



on peut facilement constater que les messages sont différents d'un sens à l'autre des échanges entre composants. cette liste de signaux nous sert de base pour la modélisation des interfaces de communication entre les composants ainsi que le présente la figure 2.8. Finalement, nous proposons un diagramme de composant pour représenter les briques architecturales de composants du protocole DataStreamX. Chaque composant a ainsi des interfaces qui lui permettent de communiquer avec les autres, mais ces seules interfaces ne suffisent pas. il manque encore des ports de connexion pour pouvoir les interconnecter ce qui justifie l'usage de ces composants présenté dans la figure 2.9.



FIGURE 2.8 – Interfaces des signaux échangés dans DataStreamX

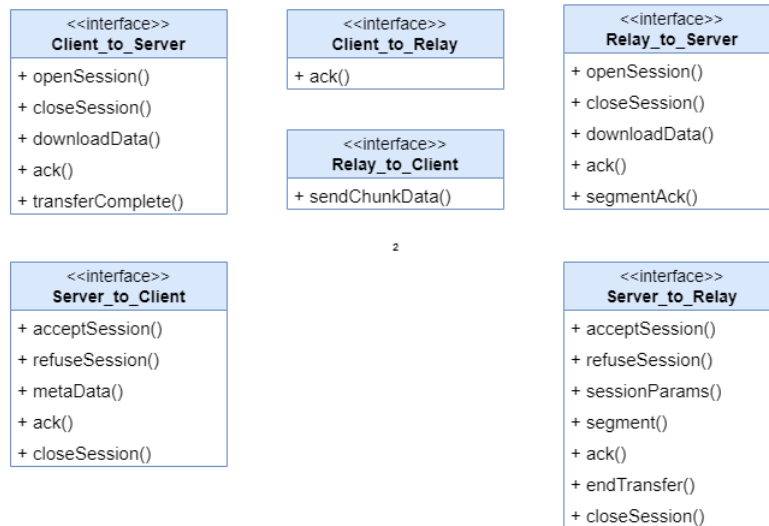
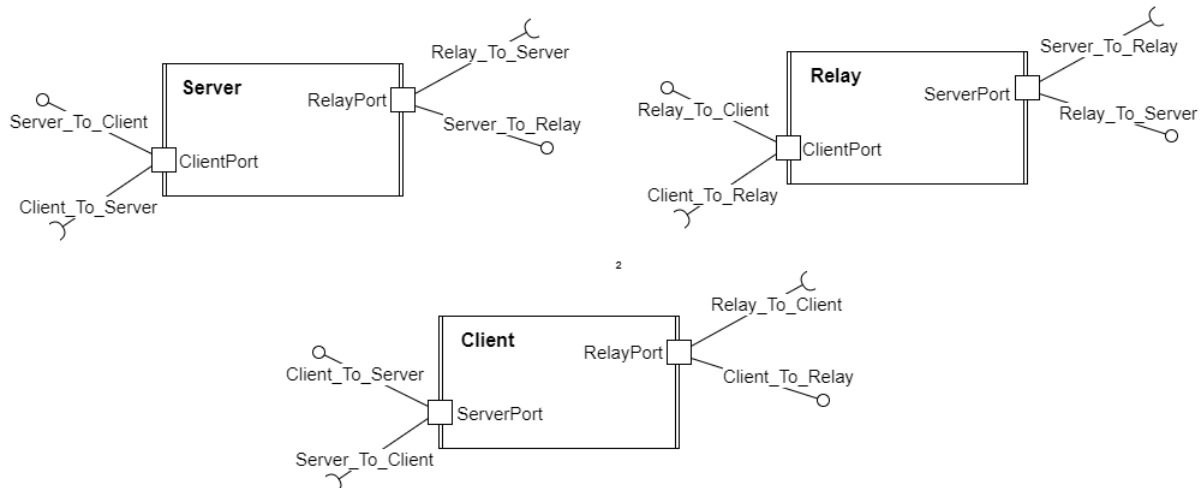


FIGURE 2.9 – Structures de Composant des entités de base de DataStreamX



La figure 2.9 présente les briques architecturales qui représentent les composants DataStreamX, on peut distinguer les interfaces que nous avons définies précédemment et les ports sur lesquels les autres composants peuvent se connecter à un composant. en réalité, dans le composant **Client**, les ports ServerPort et RelayPort sont identiques dans un cas de clonage de session, car les relais ne sont que des chemins supplémentaires vers le serveur, ils sont cependant bien distincts dans le cas d'une création de session directe entre le client et le relais.

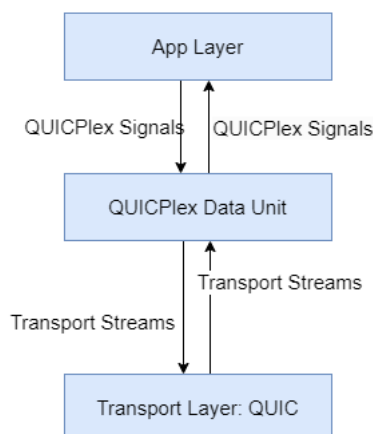




2.2.2.3 Conception Architecturale

Avant de poursuivre notre travail de conception du protocole DataStreamX, nous allons un peu moins abstraire les composants serveur, client et relais du protocole pour les unifier en une structure commune prenant part à un processus de communication, c'est-à-dire pouvant envoyer et recevoir des données. Ainsi, un composant DataStreamX peut être décomposé en plusieurs couches, une couche applicative, une couche d'encodage et décodage des paquets de données et enfin une couche transport permettant d'acheminer les données d'une extrémité à une autre. la figure 2.10 illustre la structure d'un composant DataStreamX et la figure 2.11 donne une architecture globale du protocole QUIC d'un point de vue composant, c'est-à-dire une partie émettrice de donnée et une partie réceptrice de données

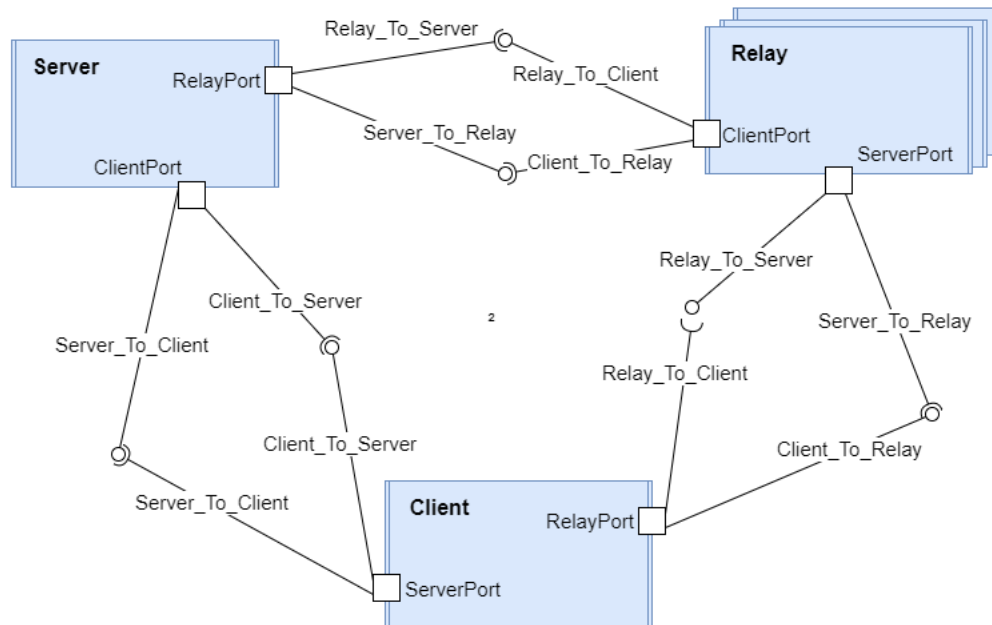
FIGURE 2.10 – Structure interne des composants DataStreamX



Dès lors que nous avons plus de détails sur la structure interne d'un composant DataStreamX, nous pouvons revenir au diagramme de structures composites afin de proposer une architecture globale du protocole dans la figure 2.13. puisque nous avons déjà défini les briques architecturales de ce diagramme, nous les associons pour ressortir un aperçu de leurs interconnexions avant de faire une conception détaillée de chaque composant.



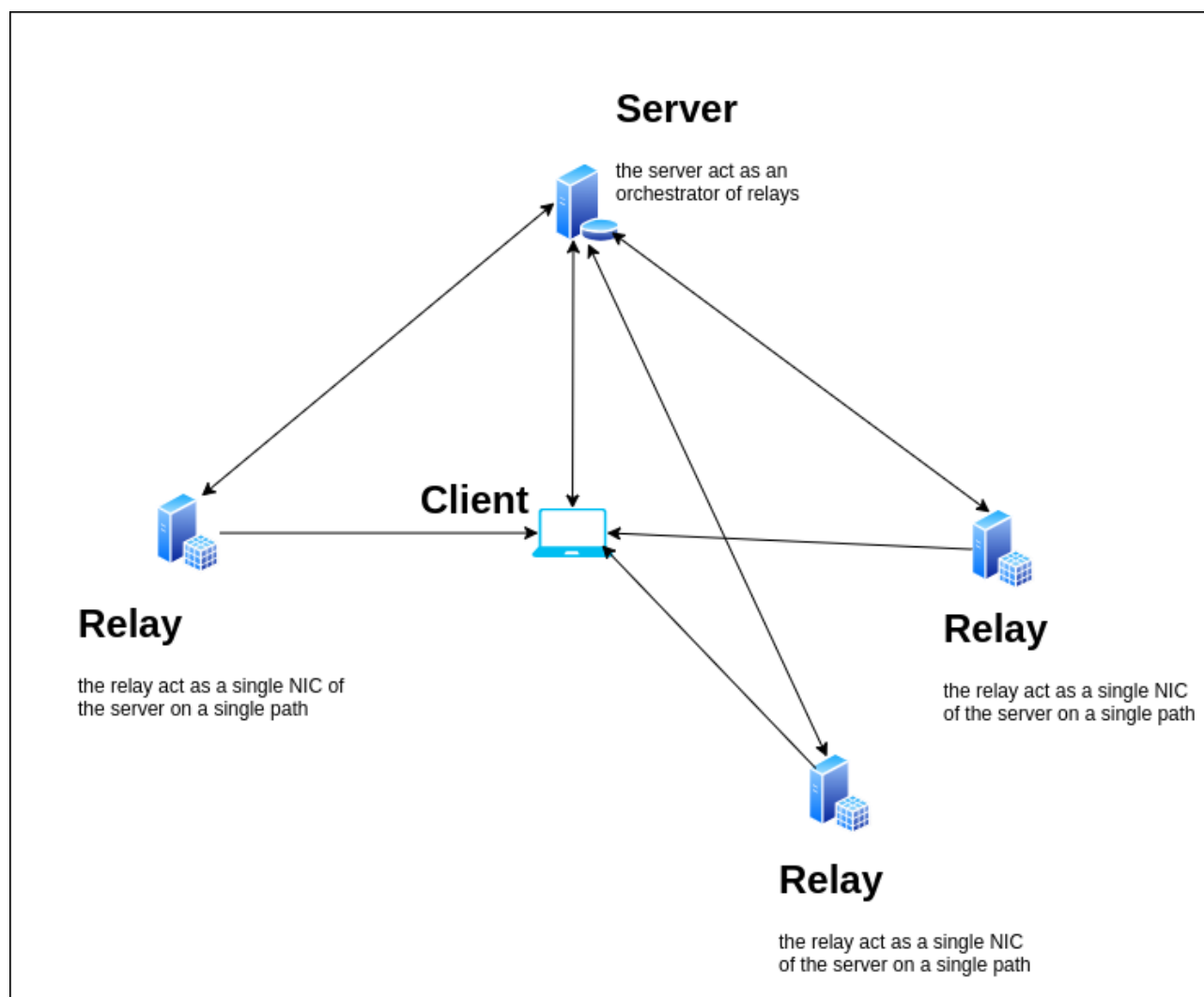
FIGURE 2.11 – Structures de Composant des entités de base de DataStreamX





2.2.3 Conception détaillée

FIGURE 2.12 – Architecture fonctionnelle du protocole DataStreamX



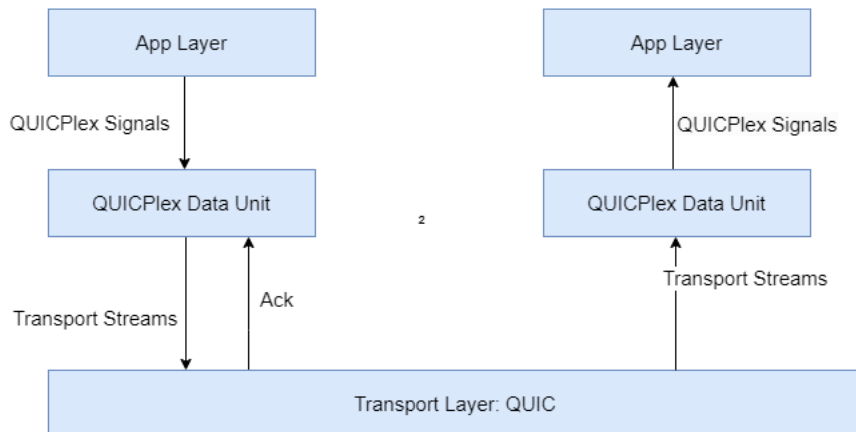
Dans cette partie, nous allons revenir sur la structure interne des composants DataStreamX et considérer les échanges entre eux comme étant de simples séquences d'envoi et de réception des messages, ceci avec pour objectif de simplifier et mieux présenter le fonctionnement du protocole. La figure **2.12** présente de manière détaillée un échange de message entre deux composants, la source et la destination peuvent être le serveur, le relai ou le client. La couche applicative sert à traiter les données transférées, il peut s'agir de la reconstitution des données ou de l'orchestration du transfert ou même la lecture de portion de données sur disque. La couche de gestion de données DataStreamX se charge de la conversion des signaux applicatifs en des paquets de données DataStreamX et inversement, en la sérialisation de ces paquets en flux de données pour la couche transport et désérialisation des flux de données issus de la couche transport en paquets





de données DataStreamX. cette couche est donc en réalité deux couches combinées, l'une pour la création des paquets soit à partir de signaux soit à partir de flux de données, l'autre pour la sérialisation des paquets et la désérialisation des paquets reçus.

FIGURE 2.13 – Structures d'un échange entre deux composants DataStreamX



2.2.3.1 Fonctionnement des composants DataStreamX

dans cette section, nous présentons les fonctionnalités propres à chaque composant et dans la partie suivante, nous allons présenter les machines à état illustrant le fonctionnement des composants du protocole dans des cas d'usages simples et généralistes tels que l'ouverture de session et l'envoi de données.

Le protocole DataStreamX repose sur trois principaux composants : le client, le serveur et les relais. Voici une description détaillée de chaque composant et leur comportement.

Le Client

- **Rôle** : Le client initie la demande de récupération des données et reçoit les segments des relais pour reconstituer le fichier final.
- **Comportement** :
 - **Requête initiale** : Le client envoie une requête de récupération de données au serveur.
 - **Réception des données** : Après sélection des relais par le serveur, le client commence à recevoir les segments de données de plusieurs relais simultanément.
 - **Vérification des données** : Chaque segment reçu est soumis à une vérification d'intégrité via un mécanisme de *checksum*.





- **Reconstitution des données** : Le client réassemble les segments pour reconstituer le fichier d'origine.
- **Accusittement** : Le client envoie des accusés de réception au serveur après la réception de chaque segment.
- **Fermeture** : Lorsque tous les segments ont été reçus et vérifiés, le client ferme la session de transfert.

Le Serveur

- **Rôle** : Le serveur est le coordinateur central du protocole, chargé de sélectionner les relais et de gérer l'acheminement des segments de données vers le client.
- **Comportement** :
 - **Attente de requêtes** : Le serveur est en attente d'une requête de récupération de données envoyée par un client.
 - **Sélection des relais** : Le serveur sélectionne les relais disposant des données en fonction de critères comme la bande passante, la latence et la disponibilité d'énergie verte.
 - **Coordination du transfert** : Il divise le fichier à transmettre en segments et envoie à chaque relais les informations nécessaires pour établir une session de transfert avec le client.
 - **Suivi des transferts** : Le serveur suit la progression du transfert, reçoit des confirmations des relais concernant l'envoi des segments et gère les retransmissions en cas de pertes.
 - **Fermeture de session** : Une fois tous les segments reçus par le client, le serveur envoie des requêtes de fin de transfert aux relais et au client.

Les Relais

- **Rôle** : Les relais jouent le rôle de médiateurs entre le serveur et le client, en stockant des segments de données et en les transmettant au client sous la supervision du serveur.
- **Comportement** :
 - **Réception des paramètres de session** : Lorsqu'un relais est sélectionné par le serveur, il reçoit les informations de session (segment à transmettre, adresse du client, etc.).
 - **Transmission des segments** : Le relais transmet le segment assigné au client.
 - **Accusittement** : Le relais envoie au serveur une confirmation une fois le segment envoyé avec succès.





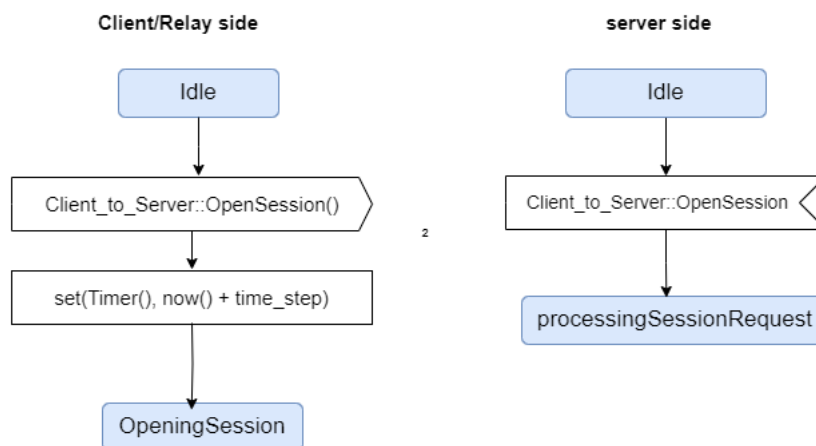
- **Gestion des échecs** : Si un relais échoue à transmettre son segment, il cesse la transmission après un délai, et le serveur peut alors redistribuer le segment à un autre relais.
- **Fermeture de session** : Une fois tous les segments transférés, le relais reçoit une requête de fin de session du serveur et ferme la session de transfert avec le client.

Les fonctionnalités suscitées peuvent se réduire en de simples opérations bien orchestrées par des algorithmes d'orchestration que nous décrivons un peu plus loin dans la section **2.4 Algorithmes d'orchestration de l'envoi des données**. Les opérations simples auxquelles sont réduites le transfert de données sont celles liées à la gestion des sessions (ouverture, acceptation, refus et fermeture) entre les composants participant au transfert et celles liées à la gestion des flux de données via le protocole de transport (écriture, lecture des flux de données) entre ces composants.

La conception détaillée de ces opérations consiste en la modélisation d'une machine à état montrant l'évolution de l'état des composants et du protocole au fur et à mesure que les signaux sont échangés entre les composants.

2.2.3.2 Ouverture de session

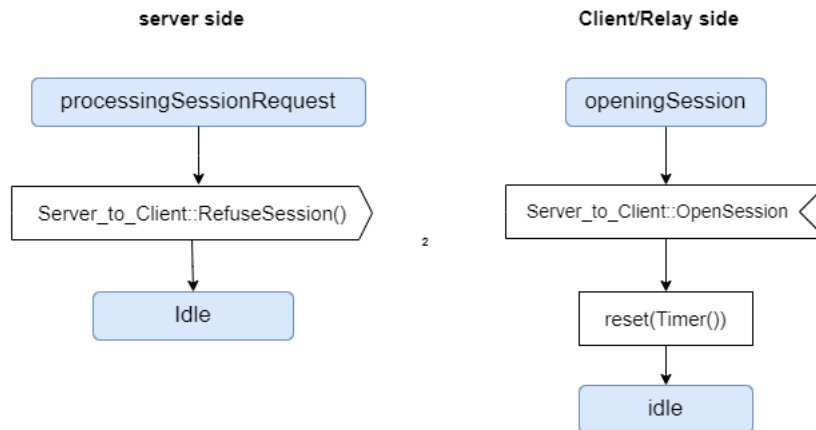
FIGURE 2.14 – *Ouverture de session*





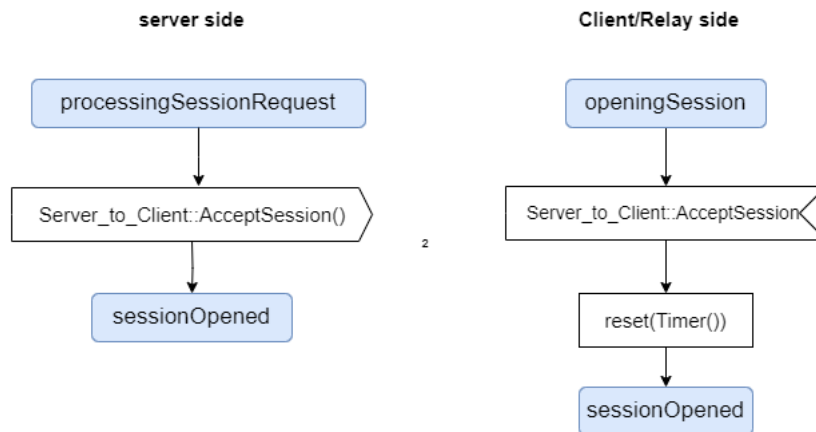
2.2.3.3 Refus ou échec de session

FIGURE 2.15 – *Refus de session*



2.2.3.4 Acceptation de session

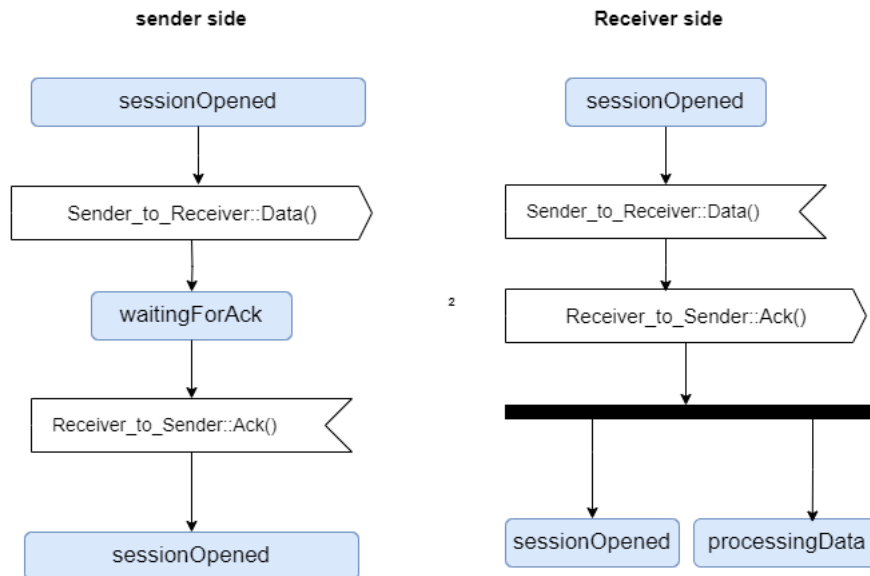
FIGURE 2.16 – *Acceptation de session*





2.2.3.5 Envoi de données

FIGURE 2.17 – *Envoi de données*



2.2.3.6 Encapsulation des signaux dans DataStreamX

Dans le protocole DataStreamX, la communication entre les entités du réseau (serveur, relais, client) repose sur l'échange de paquets structurés, qui garantissent une transmission efficace des données et une gestion flexible des connexions. Ces paquets sont divisés en deux parties : l'en-tête et le contenu (payload). Pour l'instant les signaux que nous avons définis servent principalement à identifier les paquets, mais ne renseignent pas plus sur leur contenu. Étant donné que nous nous reposons sur un protocole de transport qui garantit la livraison des paquets, nous pouvons nous simplifier la structure des entêtes de nos propres paquets à l'identification du paquet et à renseigner sur la taille de la charge utile, la charge utile du paquet contient les données utiles à transférer entre deux pairs du protocole. C'est grâce à l'article **Construire un protocole de jeu en réseau** [8], nous avons pu définir la structure de nos paquets.

Structure générale d'un paquet DataStreamX

Chaque paquet est constitué de deux éléments principaux :

- **En-tête** : Partie fixe contenant des informations essentielles pour l'identification et la gestion du paquet, notamment :
 - **PacketType** : Type de paquet (données, contrôle, accusé de réception)
 - **SequenceNumber** : Numéro de séquence unique, garantissant l'ordre des paquets.





- **PayloadLength** : Indique la taille des données transportées.
- **Payload** : Partie variable contenant les données à transmettre, dont la taille est spécifiée par l'en-tête.

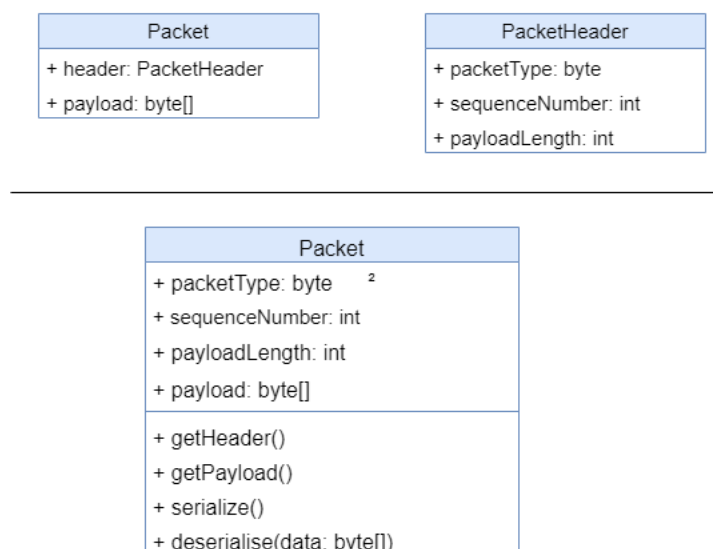
Cette structure simple permet une gestion uniforme des paquets, quel que soit leur type ou leur rôle dans le protocole.

Structure générale d'un paquet DataStreamX

Dans le protocole DataStreamX, trois principaux types de paquets sont utilisés pour orchestrer le transfert des données :

- **Paquet de données (DataPacket)** : Transporte les segments de fichiers. Le payload contient les données utilisateur, numérotées pour une reconstitution correcte au destinataire.
- **Paquet d'accusé de réception (AckPacket)** : Confirme la réception des paquets de données, avec le numéro de séquence du paquet reçu.
- **Paquet de contrôle (ControlPacket)** : Gère les connexions, la synchronisation et la fermeture de session, pour distinguer les opérations spécifiques, il existe plusieurs paquets de controle.

FIGURE 2.18 – Structures d'un paquet DataStreamX





Gestion des Paquets dans DataStreamX

Chaque type de paquet est traité différemment par le serveur central, les relais et les clients. Par exemple, lors de la réception d'un paquet de données, le client doit vérifier le SequenceNumber pour s'assurer qu'il n'y a pas de perte ou de duplication, puis traiter le payload en conséquence. Les paquets de contrôle, quant à eux, sont interprétés en fonction de leur type pour ajuster les paramètres de connexion ou gérer la distribution des segments de données.

2.2.4 Simulation et Validation

Pour Valider notre protocole, nous devons faire une simulation de son fonctionnement tel que décrit dans le diagramme d'activité de la figure 2.2. par souci de simplicité, nous allons faire une simulation avec un relais. nous considérons deux fenêtres coulissantes de taille 2.

FIGURE 2.19 – Simulation et validation de DataStreamX

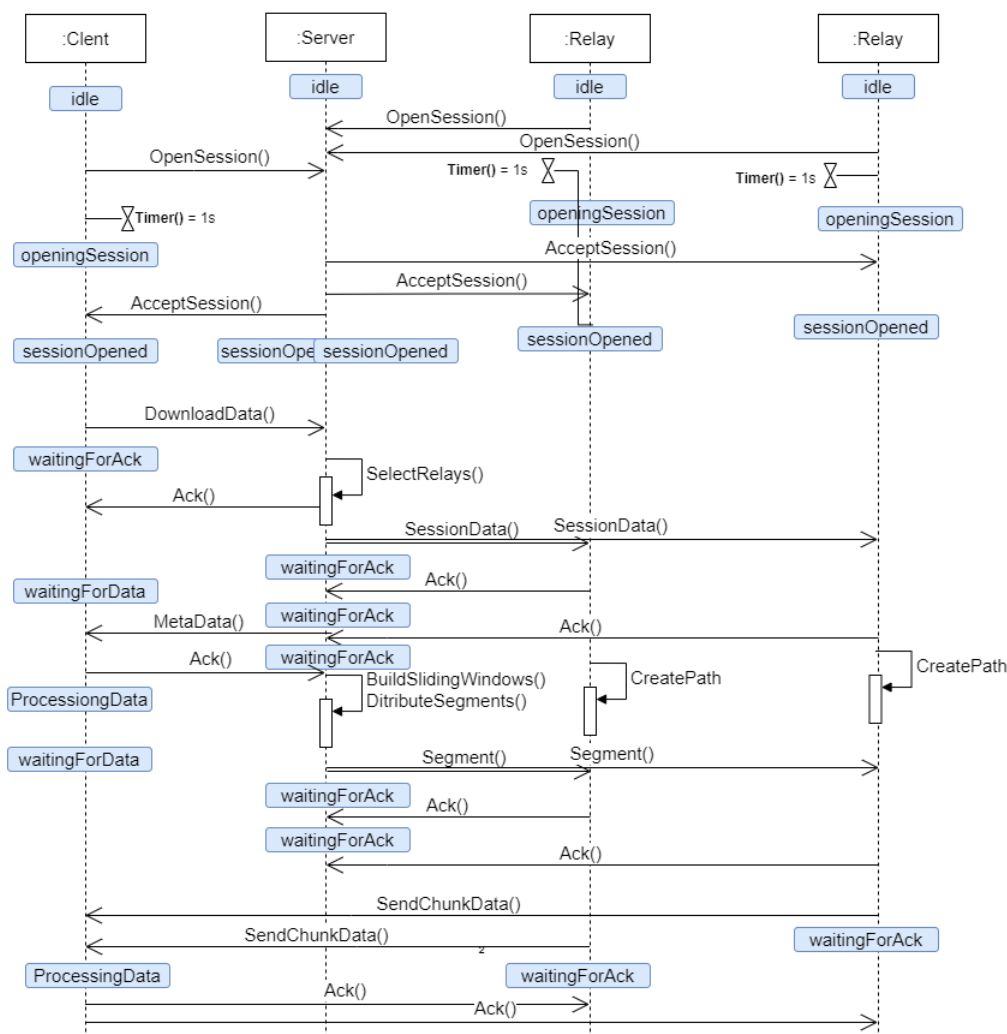
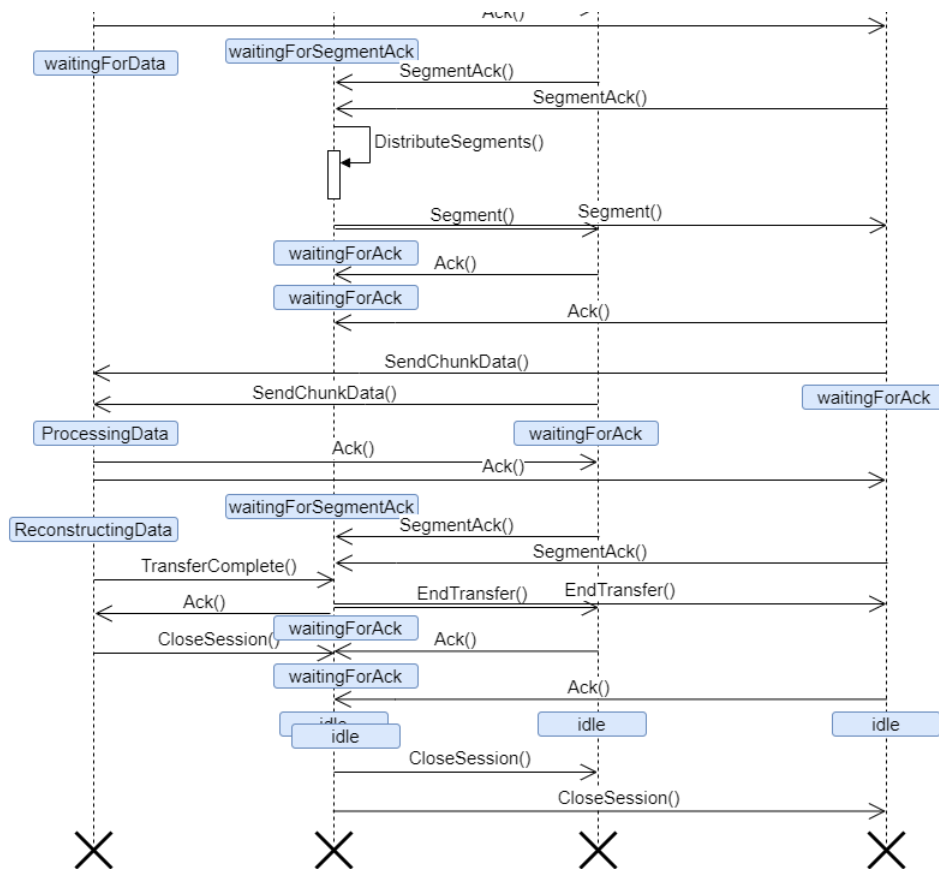




FIGURE 2.20 – *Simulation et validation de DataStreamX (suite)*



Les figures 2.18 et 2.19 illustrent un processus de transfert de données simple, les signaux sont bien envoyés et correctement interprétés, on voit également l'évolution de l'état des composants suivant conformément aux machines à état que nous avons définies dans la section 2.3.3. pour ne pas d'avantage étendre ce document, nous nous limiterons à ce diagramme pour la validation du protocole, nous précisons néanmoins que la validation du protocole a continué avec d'autres cas d'usages plus évolués.

2.3 Algorithmes d'orchestration du transfert des données

L'optimisation du transfert de données repose sur une orchestration minutieuse qui comprend le calcul des méta-paramètres statiques du transfert, la sélection optimale des relais, ainsi que la répartition optimale des segments aux relais. Cette section décrit les étapes clés et présente les algorithmes utilisés pour orchestrer l'envoi des données.





2.3.1 Calcul des méta-paramètres statiques

Avant de procéder au transfert des données, il est nécessaire de calculer les méta-paramètres statiques qui vont influencer l'organisation du transfert, tels que le nombre de fenêtres coulissantes et le nombre de portions de données par fenêtre.

2.3.1.1 Nombre de fenêtres coulissantes

Le nombre de fenêtres coulissantes dépend de la taille totale des données K à transférer et de la taille de la fenêtre de réception F . Chaque fenêtre contient des portions de données et permet de synchroniser le transfert entre le client et les relais. Le nombre total de fenêtres N_f est donné par l'expression :

$$N_f = \frac{K}{F}$$

où K est la taille des données à transférer (en octets) et F est la taille de la fenêtre de réception (en octets).

2.3.1.2 Nombre de portions de données par fenêtre coulissante

Chaque fenêtre coulissante contient plusieurs portions de données, et le nombre de portions N_p dans une fenêtre dépend de la taille de la charge utile maximale d'un paquet de données $P_{maxutil}$. La relation entre le nombre de portions et la taille de la charge utile est donnée par l'expression :

$$N_p = \frac{F}{P_{maxutil}}$$

où $P_{maxutil}$ est la taille maximale de la charge utile d'un paquet, et F est la taille de la fenêtre de réception. Ce paramètre définit combien de paquets peuvent être envoyés dans une seule fenêtre avant de recevoir des accusés de réception.

2.3.2 Sélection des relais

La sélection des relais repose sur une combinaison de la bande passante disponible et de la disponibilité de l'énergie verte, mais il est nécessaire d'éviter un algorithme qui pourrait privilégier uniquement les relais ayant une plus grande bande passante. Le but est de maximiser à la fois l'efficacité énergétique et la rapidité du transfert. plusieurs types d'algorithmes peuvent être implémenté pour résoudre ce problème comme des algorithmes, génétiques, des algorithmes à heuristique. Pour notre cas, nous avons utilisé un algorithme glouton.





2.3.2.1 Algorithme de sélection des relais

Algorithm 1 Sélection optimale des relais

Require: Liste des relais disponibles $Relais[]$, seuil de bande passante minimale BP_{min} , seuil d'énergie verte EV_{min} , nombre de relais souhaités, N_{relais}

Ensure: Liste des relais sélectionnés, $RelaisSélectionnés[]$

- 1: **Initialize** $RelaisSélectionnés \leftarrow []$
- 2: **Initialize** $BP_{total} \leftarrow 0, EV_{total} \leftarrow 0$
- 3: **for** chaque relais $R \in Relais[]$ **do**
- 4: **if** $R.BandePassante \geq BP_{min}$ **and** $R.EnergieVerte == true$ **then**
- 5: $RelaisSélectionnés \leftarrow RelaisSélectionnés \cup R$
- 6: $BP_{total} \leftarrow BP_{total} + R.BandePassante$
- 7: $EV_{total} \leftarrow EV_{total} + 1$
- 8: **if** $BP_{total} \geq BP_{min}$ **and** $EV_{total} \geq EV_{min}$ **and** $|RelaisSélectionnés| \geq N_{relais}$ **then**
- 9: **return** $RelaisSélectionnés$
- 10: **else**
- 11: Ajuster les seuils et rechercher d'autres relais

Cet algorithme sélectionne les relais en fonction de leur bande passante et de la disponibilité de l'énergie verte. Les relais sont choisis de manière à maximiser l'utilisation des relais verts tout en garantissant une bande passante totale suffisante.





2.3.3 Répartition optimale des segments

Une fois les relais sélectionnés, il est nécessaire d'attribuer les segments de données de manière optimale afin de maximiser l'utilisation de la bande passante et de l'énergie verte. L'attribution des segments prend en compte la bande passante et un facteur de pondération lié à la disponibilité de l'énergie verte.

2.3.3.1 Algorithme de répartition des segments

Algorithm 2 Répartition optimale des segments avec prise en compte de l'énergie verte

Require: Taille totale des données K , liste des relais sélectionnés $RelaisSélectionnés[]$, facteur d'énergie $alpha_{energie}[]$

Ensure: Attribution des segments $Segments[]$

```

1: Initialize  $BP_{total} \leftarrow \sum_{i=1}^{|RelaisSélectionnés|} (RelaisSélectionnés[i].BandePassante \times$ 
    $alpha_{energie}[i])$ 
2: Initialize  $Allocation \leftarrow []$ ,  $début\_segment \leftarrow 0$ 
3: for chaque relais  $R \in RelaisSélectionnés[]$  do
4:    $Score_R \leftarrow R.BandePassante \times alpha_{energie}[R]$ 
5:    $P_i \leftarrow \left( \frac{Score_R}{BP_{total}} \right) \times K$ 
6:    $P_i \leftarrow \text{floor}(P_i)$ 
7:    $Allocation \leftarrow Allocation \cup P_i$ 
8:  $total\_allocation \leftarrow \sum Allocation$ 
9:  $delta \leftarrow K - total\_allocation$ 
10: while  $delta > 0$  do
11:   for chaque  $i \in RelaisSélectionnés[]$  do
12:     if  $delta == 0$  then
13:       break
14:      $Allocation[i] \leftarrow Allocation[i] + 1$ 
15:      $delta \leftarrow delta - 1$ 
16:  $Segments \leftarrow []$ 
17: for chaque  $i \in RelaisSélectionnés[]$  do
18:    $fin\_segment \leftarrow début\_segment + Allocation[i] - 1$ 
19:    $Segments \leftarrow Segments \cup [début\_segment, fin\_segment]$ 
20:    $début\_segment \leftarrow fin\_segment + 1$ 
   return  $Segments$ 

```

Cet algorithme répartit les segments de données proportionnellement à la bande passante et à l'énergie verte disponible de chaque relais. La répartition est ajustée pour s'assurer que la somme des parts attribuées égale exactement la taille totale des données K .





2.3.3.2 Ajustement des parts pour minimiser le temps de transfert

Après la répartition initiale, si certains relais terminent leur transfert beaucoup plus tôt que d'autres, il est nécessaire de réajuster les segments restants en les attribuant aux relais non verts pour minimiser le temps de transfert total. Ce processus est effectué sans tenir compte de la disponibilité d'énergie verte, mais uniquement en fonction de la bande passante.

Algorithm 3 Ajustement des parts pour minimiser le temps de transfert

Require: Temps de transfert initial $TempsInitial[]$, relais sans énergie verte $RelaisNonVerts[]$, segments restants $K_{restant}$

Ensure: Nouvelle attribution des segments $NouvelleAttribution[]$

- 1: **Initialize** $T_{max} \leftarrow \max(TempsInitial)$
- 2: **Initialize** $SegmentsRestants \leftarrow []$
- 3: **for** chaque $R \in RelaisNonVerts[]$ **do**
- 4: **if** $R.Temps > T_{max}$ **then**
- 5: $SegmentsRestants \leftarrow SegmentsRestants \cup Segments[R]$
- 6: $BP_{total_non_vert} \leftarrow \sum_{i=1}^{|RelaisNonVerts|} RelaisNonVerts[i].BandePassante$
- 7: **for** chaque $S \in SegmentsRestants$ **do**
- 8: **for** chaque $R \in RelaisNonVerts[]$ **do**
- 9: $P_i \leftarrow \left(\frac{R.BandePassante}{BP_{total_non_vert}} \right) \times K_{restant}$
- 10: $P_i \leftarrow \text{floor}(P_i)$
- 11: $NouvelleAttribution \leftarrow NouvelleAttribution \cup (S, R)$

return $NouvelleAttribution$

Cet ajustement assure que les relais sans énergie verte participent davantage au transfert pour réduire le temps total de transfert, tout en réattribuant les segments restants de manière optimale.

Bilan du Chapitre

En résumé, ce chapitre a permis de définir clairement les objectifs et le fonctionnement du protocole DataStreamX. Nous avons fait une modélisation mathématique des objectifs poursuivis par DataStreamX et avons montré qu'atteindre ces objectifs revenait à résoudre un problème d'optimisation multicritères. Nous avons également fait la spécification des besoins de DataStreamX et posé les bases conceptuelles du protocole conformément aux principes de l'ingénierie des protocoles. Enfin, nous avons présenté notre approche pour atteindre les objectifs visés par DataStreamX à travers les différents algorithmes permettant l'orchestration du transfert des données, c'est ainsi que nous avons illustré le principe de fenêtre coulissant et définis des algorithmes adaptatifs de sélection des relais et de répartition des segments de données sous les contraintes de bande passante et de disponibilité d'énergie renouvelable.



IMPLÉMENTATION ET RÉSULTATS

C Ce dernier chapitre est consacré à l'implémentation du **protocole DataStreamX** et à l'évaluation des résultats obtenus. Nous décrirons les technologies et outils utilisés pour développer le protocole, ainsi que l'**environnement de test** mis en place pour valider son efficacité. Les résultats expérimentaux, notamment en termes de **débit**, **latence** et **réduction de la consommation énergétique**, seront analysés et comparés aux solutions de transfert traditionnelles. Nous concluons en discutant des bénéfices apportés par DataStreamX, ainsi que des **perspectives d'amélioration future**.

Sommaire

2.1	Rappel du problème et des objectifs poursuivis	27
2.1.1	Objectif général	27
2.1.2	Modélisation mathématique du problème d'optimisation	27
2.1.3	Contraintes du problème	28
2.1.4	Complexité du problème	29
2.2	Modélisation UML du protocole DataStreamX	30
2.2.1	Spécification des exigences	30
2.2.2	Analyse	31
2.2.3	Conception détaillée	43
2.2.4	Simulation et Validation	50
2.3	Algorithmes d'orchestration du transfert des données	51
2.3.1	Calcul des méta-paramètres statiques	52
2.3.2	Sélection des relais	52
2.3.3	Répartition optimale des segments	54



3.1 Choix des technologies et des outils

Dans cette section, nous présentons les langages de programmation, les frameworks et les outils que nous avons utilisé pour le développement et la mise en place de notre protocole de transfert de données.

3.1.1 Les langages de programmation et les bibliothèques

3.1.1.1 Go et MP-QUIC

Nos travaux sont basé sur l'implémentation de MP-QUIC que nous avons amélioré pour répondre a nos besoins au niveau du protocole de transport. L'implémentation en Go du protocole MP-QUIC a donc orienté notre usage du langage de programmation Go encore appelé Golang. Nous avons donc implémenté notre protocole de transfert de données au dessus de MP-QUIC en utilisant ce langage. Il est reconnu pour sa gestion efficace de la concurrence grâce à l'utilisation des goroutines, qui sont des threads légers gérés par son runtime. Cela permet une meilleure performance lors de la gestion de multiples connexions réseau simultanées, ce qui est crucial pour des transferts de données massifs.

3.1.1.2 Python et Simpy

Nous avons utilisé Python pour l'implémentation du simulateur nécessaire pour tester la performance (technique et énergétique) des algorithmes de synchronisation grâce a la bibliothèque Simpy. Simpy est une bibliothèque Python conçue pour la simulation de processus concurrents, tels que ceux trouvés dans des réseaux distribués, ce qui en fait un outil idéal pour simuler les transferts de données parallèles. En utilisant Simpy, il a été possible de recréer des scénarios de test complexes, intégrant des variables comme la latence, la bande passante, et la congestion, afin d'évaluer de manière précise la performance de notre protocole dans des conditions réalistes.

3.1.1.3 Bash

Nous avons utilisé des cripts Bash pour automatiser la création de réseaux virtuels, les tests et les déploiements de la solution. Ces scripts nous ont servi à orchestrer l'exécution des simulations, collecter les résultats, et gérer les différentes configurations des environnements de test. Cela a permis de rationaliser le processus de test et d'assurer la reproductibilité des résultats, facilitant ainsi l'analyse des performances sous différents scénarios.

3.1.2 Les outils utilisés

3.1.2.1 Visual Studio Code (VS Code)

Visual Studio Code est un éditeur de code source léger mais puissant, prenant en charge plusieurs langages de programmation. Dans ce projet, VS Code a été utilisé pour l'écriture, le





débogage, et l'exécution du code, ainsi que pour la gestion des extensions liées à Docker et Git, contribuant ainsi au développement et à la gestion du projet.

3.1.2.2 Git et GitHub

Git est un système de contrôle de version distribué, et GitHub est une plateforme d'hébergement de projets utilisant Git. Ces outils ont été essentiels pour le suivi des versions du code source, la collaboration entre les membres de l'équipe, et la gestion des branches de développement. GitHub a également été utilisé pour héberger le code source et faciliter les révisions de code.

3.1.2.3 Docker

Docker est une plateforme permettant de créer, déployer, et exécuter des applications dans des conteneurs. Dans ce projet, Docker a été utilisé pour containeriser les différentes parties de l'application, assurant ainsi un environnement cohérent et reproductible pour les tests et le déploiement, tout en isolant les dépendances pour chaque service. Il nous a également servi de support pour la création d'un réseau permettant de connecter nos différents conteneurs entre eux pour les essais.

3.1.2.4 Mininet

Mininet est un émulateur de réseau virtuel qui permet de créer un réseau complet d'hôtes, de commutateurs, de routeurs, et de liens sur une seule machine. Cet outil a été utilisé pour simuler un réseau et tester le protocole de communication développés dans ce projet, permettant une validation rapide des concepts sans nécessiter de matériel réseau réel.

3.1.2.5 Wireshark

Wireshark est un analyseur de protocoles réseau qui capture et examine les paquets de données circulant sur un réseau. Cet outil a été utilisé pour diagnostiquer les problèmes de communication et analyser les performances du protocole.

3.1.2.6 Draw.io

Draw.io est un outil en ligne qui permet de créer des diagrammes variés, comme des diagrammes de flux, des organigrammes, et des diagrammes de séquence. Cet outil a été utilisé pour concevoir et visualiser les diagrammes de séquence, les architectures des composants et la structure du protocole conçu, facilitant ainsi la modélisation des interactions entre les différents composants.





3.1.2.7 Inria Notes

Inria Notes est un outil de prise de notes développé par Inria, permettant de structurer et d'organiser des idées de manière efficace. Dans ce projet, Inria Notes a servi à prendre des notes détaillées sur les concepts théoriques, les idées de conception, de développement, et les réunions, ce qui a permis de garder une trace cohérente et structurée des différentes phases du projet.

3.2 Implémentation du Protocole

L'implémentation de notre protocole a suivi une approche modulaire et orientée objet, permettant de structurer le code de manière claire et maintenable tout en assurant une flexibilité pour de futures extensions. Chaque composant du protocole a été soigneusement conçu pour répondre à des exigences spécifiques tout en permettant une intégration fluide avec les autres modules. Cette section décrit les principales composantes de l'implémentation, notamment la gestion des connexions multiplexées, le calcul adaptatif des fenêtres de transfert, et la répartition des segments de données.

3.2.1 Architecture Modulaire

L'architecture du protocole repose sur une modularité stricte, où chaque composant du protocole est responsable d'une tâche bien définie. Cette approche facilite la gestion des différentes fonctionnalités, améliore la lisibilité du code et simplifie la maintenance et l'extension du protocole. Les principaux modules sont :

- **Gestion des Connexions** : Ce module est responsable de l'établissement, de la gestion et de la fermeture des connexions entre le serveur central, les relais et les clients. En exploitant les capacités de QUIC, le protocole supporte le multiplexage des connexions, permettant ainsi à plusieurs flux de données de circuler simultanément sur une même connexion.
- **Calcul des Fenêtres de Transfert** : Ce module calcule les fenêtres de transfert des fichiers, segmentant chaque fichier en morceaux de taille prédéfinie. Chaque fenêtre est ensuite subdivisée en segments destinés à différents relais, selon un algorithme adaptatif qui prend en compte la latence et le débit des relais pour optimiser la distribution des données.
- **Répartition des Segments** : Une fois les fenêtres et segments calculés, ce module se charge de leur répartition entre les différents relais. Le serveur central envoie les segments correspondants à chaque relais, en tenant compte des performances de chaque connexion pour ajuster dynamiquement la distribution lors des transferts suivants.
- **Gestion des Paquets** : Ce module s'occupe de la création, de la sérialisation, et de la désérialisation des paquets échangés entre les différentes entités du protocole. Il gère





également les différents types de paquets, incluant les paquets de données, les acquittements (ACKs), et les paquets de contrôle. Une attention particulière a été portée à la taille et à la structure des paquets pour garantir qu'ils respectent la taille maximale de 1Ko, conformément à notre conception.

3.2.2 Gestion des Connexions Multiplexées

La gestion des connexions multiplexées héritée de QUIC est un aspect fondamental de notre protocole, permettant à plusieurs flux de données d'être gérés simultanément sur une même connexion. En utilisant l'implémentation **QUIC-go** du protocole QUIC, chaque connexion est capable de transporter plusieurs flux, chacun représentant un segment de fichier ou une autre unité de données. Cela permet au protocole de maximiser l'utilisation de la bande passante disponible et de minimiser les temps d'attente, même lorsque plusieurs fichiers sont transférés en parallèle.

La concurrence native de Golang a été exploitée pour gérer ces connexions de manière efficace. Chaque flux est traité dans une goroutine séparée, permettant au serveur de gérer plusieurs connexions simultanément sans engendrer de goulots d'étranglement. De plus, les connexions avec chaque relais sont également gérées en parallèle, ce qui assure que les segments de fichier sont distribués rapidement et efficacement à travers le réseau.

3.2.3 Calcul Adaptatif des Fenêtres et des Segments

Notre protocole utilise un mécanisme de calcul adaptatif pour définir les fenêtres de transfert et répartir les segments de fichier. Le serveur central commence par déterminer la taille totale du fichier à transférer, puis segmente ce fichier en plusieurs fenêtres, chaque fenêtre correspondant à une unité de transfert.

Ensuite, chaque fenêtre est elle-même subdivisée en segments, chacun étant assigné à un relais spécifique. Cette répartition est dynamique et basée sur les performances observées de chaque relais, notamment en termes de RTT (Round-Trip Time) et de taux de retransmission. Les relais les plus performants se voient attribuer des segments plus importants, tandis que les relais plus lents ou moins fiables reçoivent des segments plus petits, permettant ainsi d'optimiser le temps total de transfert.

3.2.4 Répartition Dynamique des Segments de Données

Une fois les fenêtres et les segments définis, le serveur central procède à l'envoi de ces segments vers les relais. Ce processus se fait en parallèle, avec des connexions séparées pour chaque relais, optimisant ainsi l'utilisation de la bande passante disponible.

Le serveur utilise les statistiques de transfert des fenêtres précédentes pour ajuster la répartition des segments dans les fenêtres suivantes. Par exemple, si un relais particulier affiche





un temps de réponse plus rapide, le serveur pourra lui attribuer une plus grande portion des segments lors du prochain cycle de transfert. Ce mécanisme adaptatif permet d'améliorer progressivement l'efficacité du transfert au fur et à mesure de son déroulement.

3.2.5 Gestion des Paquets

La gestion des paquets est cruciale pour assurer la fiabilité et la cohérence des échanges entre les différentes entités du protocole. Chaque paquet est composé d'une en-tête et d'une charge utile, avec un champ spécifique pour identifier le type de paquet (données, acquittement, contrôle) et un sous-type optionnel pour les paquets de contrôle.

Les paquets sont sérialisés avant d'être envoyés sur le réseau, en s'assurant que leur taille totale ne dépasse pas la limite de 1Ko, comme définie dans notre conception. À la réception, les paquets sont désérialisés et leur contenu est interprété en fonction du type et du sous-type du paquet. Ce processus garantit une communication claire et structurée entre le serveur, les relais, et les clients, et facilite la gestion des états de connexion.

La modularité de la gestion des paquets permet d'étendre facilement le protocole pour inclure de nouveaux types de paquets ou pour ajuster le comportement en fonction des besoins spécifiques de l'application. Cela rend le protocole non seulement performant mais aussi adaptable à divers scénarios d'utilisation.

3.2.6 Aspect énergétique

L'efficacité énergétique a été un axe central de nos travaux, particulièrement en raison de l'empreinte écologique croissante des centres de données. Pour évaluer la consommation d'énergie de manière précise, nous avons développé un simulateur capable de mesurer l'énergie utilisée par les serveurs d'un centre de données durant l'exécution des tâches, notamment les relais impliqués dans les transferts de fichiers volumineux. Ce simulateur a été utilisé dans plusieurs phases du développement de notre protocole, nous permettant ainsi d'obtenir des données fiables sur la consommation énergétique en fonction des différentes configurations de relais et des algorithmes de synchronisation de flux. Toutefois, un défi majeur est lié à l'intermittence des sources d'énergie renouvelable. Les relais alimentés par ces sources ne sont pas toujours disponibles à pleine capacité, ce qui complique la planification des transferts. Il devient donc crucial de gérer dynamiquement l'affectation des tâches de transfert en fonction des variations de disponibilité de l'énergie renouvelable. Le protocole a été conçu pour prendre en compte ces fluctuations en ajustant intelligemment la répartition des segments entre les relais, favorisant ceux alimentés par l'énergie verte lorsque celle-ci est disponible, mais prévoyant également des solutions de repli pour assurer la continuité des transferts via des relais moins écoresponsables en cas de pénurie d'énergie renouvelable. Malgré cette difficulté, les résultats montrent que la sélection judicieuse des relais permet de réduire considérablement l'empreinte énergétique des transferts tout en maintenant des performances élevées.





3.3 Environnement de simulation

Dans cette section, nous exposons les détails de conception de notre simulateur. ce simulateur a permis de mesurer les performance des algorithmes de synchronisation que nous avons développé. L'objectif de ce simulateur est de permettre d'avoir une approximation qui se rapproche le plus possible de la réalité de la consommation énergétique et de l'usage des ressources des relais au bout de l'envoi complet des segments qui leur sont attribuées à chaque déplacement de la fenêtre coulissante. Il convient de rappeler à ce niveau que la synchronisation des relais s'apparente au placement de taches où il faut placer les taches de transfert segments de données de maniere optimale et ordonnée sur les machines participant au transfert afin d'atteindre un objectif de performance énergétique bien défini sans trop dégrader les performances du transfert.

3.3.1 Le serveur

D'après [9], la puissance consommée par une machine durant son fonctionnement est répartie comme suit 43% CPU, 12% RAM, 4% Disque, le reste se reparti entre les peripheriques, la carte mère et les autres composantes. Une machine pour son fonctionnement normal consomme constante P_{static} et durant l'exécution des tâches elle consomme une puissance $P_{dynamic}$ qui dépend de l'utilisation du CPU, de la RAM ou du disque dans notre cas. Nous modélisons donc un serveur comme étant un ensemble d'unité consommant de l'énergie avec une puissance statique pour assurer son fonctionnement

- **P_{static}**
- **CPU**
- **RAM**
- **DISK**

Puisque notre travail s'effectue dans un environnement parallèle et distribué, le CPU lui même est considéré comme étant un objet possédant un nombre de coeur, une puissance minimale de fonctionnement (cette puissance sera soustraite a la puissance statique du serveur car elle y est déjà comprise) et une puissance maximale atteinte lorsque tous les coeurs sont utilisé. Dans cette modélisation, nous considererons qu'un coeur utilisé fonctionne a puissance maximale et un coeur non utilisé a puissance minimale, avec ces deux puissances homogènes pour tous les coeurs. ce sont ces puissances minimale et maximales par coeur qui permettent de calculer la puissance minimale et maximale de fonctionnement et d'un CPU. Le CPU est donc caractérisé par :

- **Puissance minimale** (P_{min} en volt)
- **Puissance maximale** (P_{max} en volt)
- **nombre de coeurs** (N sans unité)





L'usage de K ($K \leq N$) unité de processeur pendant une durée quelconque de temps T engendre une consommation énergétique donnée par :

$$E_{dyn} = T \times ((N - K) \times P_{min} + K \times P_{max})$$

durant ce temps la machine a fonctionné normalement et a consommé une énergie statique donnée par :

$$E_{static} = (T + L) \times P_{static}$$

où L représente la latence du réseau. Puisque le fonctionnement normal de la machine inclut le fonctionnement de tous les cœurs du processeur à puissance minimale, ceci réduit le calcul de l'énergie dynamique à la seule augmentation due aux cœurs actifs, soit alors :

$$E_{dyn} = T \times K \times (P_{max} - P_{min}).$$

soit une énergie totale donnée par :

$$E = (T + L) \times P_{static} + T \times K \times (P_{max} - P_{min}) + E_r + E_d.$$

Les énergies E_r et E_d sont respectivement les énergies consommées par l'usage de la RAM et du disque et seront explicitées plus bas.

Pour la RAM, Nous la représentons par sa bande passante et sa puissance instantanée, ces informations sont fournies par le fabricant :

- **Bande passante** (B en MB/s)
- **Puissance instantanée** (P en watt)

Le temps d'usage réel de la RAM est le temps nécessaire pour consommer une taille D de celle-ci et est donné par :

$$T = \frac{D}{B}$$

ainsi l'énergie consommée par celle-ci pendant ce temps est donnée par :

$$E = P \times T$$

Similairement, un disque est caractérisé par son débit et sa puissance instantanée fournis par le fabricant.

- **Débit (R)** (R en MB/s)
- **Puissance instantanée** (P en watt)

comme précédemment, le temps et l'énergie consommés pour lire une donnée de taille D sur disque sont donnés par :

$$T = \frac{D}{R} \text{ et } E = P \times T$$





3.3.2 Les tâche

Une tâche représente l'usage des ressources d'un serveur pendant une durée de temps t dans notre cas, la tâche est l'occupation du CPU, l'usage de la RAM pour bufferiser les requêtes entrantes, la lecture sur disque de les portions sur disques du fichier pendant la durée totale d'envoi des parts attribuées à un relai à chaque déplacement de la fenêtre coulissante. une tâche se caractérise donc par :

- **Durée d'exécution** (T en s)
- **Nombre d'unité CPU utilisées (coeurs)**
- **Taille de la RAM nécessaire** (D_r en MB)
- **Espace disque sollicité** (D_d en MB)

L'énergie consommée par une machine exécutant cette tâche se calcule facilement avec la formule fournie plus haut dans la modélisation du serveur

3.3.3 Le réseau

Par souci de simplicité, Nous modélisons les caractéristiques du réseau dans notre simulateur par la latence. La **latence** est ici le temps passé entre l'envoi et la réception d'une donnée à travers le réseau. Nous assumons que nous disposons d'une bande passante infinie, ceci nous permet d'alléger le calcul de l'énergie consommée par le serveur relai.

3.4 Résultats des test et simulation du Protocole

Pour évaluer les performances de notre protocole, nous avons effectué une série de tests en utilisant un benchmark représentatif des scénarios d'utilisation pour lesquels le protocole a été conçu. Ces tests visent à mesurer l'efficacité du transfert de données, la robustesse des connexions en présence de latence et de pertes de paquets, ainsi que la capacité du protocole à maintenir une performance stable sous différentes conditions de charge.

3.4.1 Configuration du Benchmark

Le benchmark choisi pour évaluer notre protocole est basé sur un scénario de transfert de fichiers volumineux dans un réseau distribué comprenant un serveur central, plusieurs relais, et un ou plusieurs clients. Ce scénario reflète une situation courante dans les systèmes de distribution de contenu ou de sauvegarde en cloud. La configuration de test est la suivante :

- **Taille des fichiers transférés** : 100 MB et 1 GB.
- **Nombre de relais** : 3 relais distribués géographiquement pour simuler la latence.
- **Clients** : 2 clients connectés simultanément, chacun recevant des segments de fichier via les relais.





- **Protocole de transport** : MP-QUIC, avec une taille de fenêtre de congestion initiale de 16 Ko et une taille de paquet de 1024 octets (1 Ko).
- **Latence réseau simulée** : 50 ms, 100 ms, et 200 ms RTT (aller-retour).
- **Taux de perte de paquets simulé** : 0%, 1%, et 5%.

3.4.2 Indicateurs de Performance

Les performances du protocole ont été mesurées en utilisant les indicateurs suivants :

- **Débit effectif (Throughput)** : Quantité de données transférées par unité de temps (en Mbps).
- **Temps de transfert total** : Durée totale nécessaire pour transférer les fichiers de 100 MB et 1 GB aux clients.
- **Efficacité des retransmissions** : Nombre de paquets retransmis en raison de pertes ou d'erreurs, comparé au nombre total de paquets envoyés.
- **Robustesse sous charge** : Performance du protocole en présence de plusieurs connexions simultanées et d'une latence accrue.

3.4.3 Résultats du Benchmark

Les résultats obtenus montrent que le protocole offre des performances compétitives et robustes, même dans des conditions de réseau dégradées. Voici les principaux résultats :

- **Débit Effectif** :
 - Pour un fichier de 100 MB, le débit moyen observé est de 85 Mbps avec une latence de 50 ms, diminuant progressivement à 75 Mbps avec une latence de 200 ms.
 - Pour un fichier de 1 GB, le débit moyen est de 82 Mbps à 50 ms de latence, et 70 Mbps à 200 ms.
- **Temps de Transfert Total** :
 - Le transfert du fichier de 100 MB a pris en moyenne 9,4 secondes avec une latence de 50 ms, et 10,7 secondes à 200 ms.
 - Le fichier de 1 GB a été transféré en 98 secondes avec une latence de 50 ms, et 112 secondes à 200 ms.
- **Efficacité des Retransmissions** :
 - Avec un taux de perte de 1%, le protocole a montré un taux de retransmission de 2,5% des paquets, ce qui reste raisonnable. À 5% de perte, le taux de retransmission a augmenté à 12%, impactant légèrement le débit effectif mais sans provoquer de chute drastique de performance.
- **Robustesse sous Charge** :





- Avec deux clients téléchargeant simultanément via trois relais, le protocole a maintenu un débit moyen combiné de 70 Mbps avec une latence de 100 ms et un taux de perte de 1%, démontrant une capacité à gérer efficacement plusieurs flux de données concurrents.

TABLE 3.1 – Résultats du Benchmark

Condition de Test	Débit Effectif (Mbps)	Temps de Transfert Total	Taux de Retransmission
Fichier 100 MB, 50 ms	85	9,4 s	2,5% (1% perte)
Fichier 100 MB, 200 ms	75	10,7 s	12% (5% perte)
Fichier 1 GB, 50 ms	82	98 s	2,5% (1% perte)
Fichier 1 GB, 200 ms	72	112 s	12% (5% perte)
Deux clients, 100 ms	70	N/A	2,5% (1% perte)

3.4.4 Comparaison par rapport aux autres protocoles existants

DataStreamX se distingue par son intégration de critères d'énergie verte dans la sélection des relais, optimisant l'empreinte carbone du transfert. Aucun des autres protocoles concurrents n'intègre cette dimension dans leurs mécanismes de transfert, la comparaison d'avec les autres protocoles est consignée dans le tableau 3.2.

3.4.5 Analyse des Résultats

Les résultats obtenus montrent que le protocole proposé répond aux exigences de performance attendues dans un environnement distribué, en particulier pour les applications nécessitant un transfert fiable et rapide de données volumineuses. Le protocole a montré une capacité à maintenir un débit élevé même en présence de conditions de réseau défavorables, telles que la latence et la perte de paquets. La modularité de notre protocole, associée à l'utilisation du protocole de transport QUIC, lui permet de s'adapter dynamiquement aux variations des conditions réseau, ce qui se traduit par une stabilité des performances sous charge.

En conclusion, les tests de performance montrent que notre protocole est bien adapté pour les scénarios de transfert de données dans des environnements distribués, offrant à la fois efficacité et robustesse. Les ajustements dynamiques des segments de fichier et la gestion des retransmissions contribuent à maintenir une expérience utilisateur fluide, même dans des conditions de réseau difficiles.



TABLE 3.2 – Comparaison des protocoles de transfert de données existant à QuicPlex

Caractéristiques	DataStream	XGridFTP	BitTorrent	FDT	MDTMFTP	HTTP/2	FTP
Type de protocole	Décentralisé, écoénergétique	FTP optimisé pour les grilles	P2P (Pair-à-Pair)	Optimisé pour transfert à haut débit	FTP amélioré pour réseaux larges	Protocole d'application (HTTP)	Protocole d'application
Multi-plexage	Oui, via des relais multiples	Oui, via connexions multiples	Oui, via les pairs	Oui, plusieurs flux TCP	Oui, plusieurs flux FTP	Oui, via une seule connexion TCP	Non
Transfert P2P	Partiellement (avec relais)	Non	Oui	Non	Non	Non	Non
Bande passante	Optimisée selon les relais	Optimisée avec plusieurs connexions	Partagée entre pairs	Maximisée via flux multiples	Maximisée via flux multiples	Optimisée par la gestion des flux	Fixe, sans optimisation
Tolérance aux pannes	Réallocation dynamique des segments	Oui, reprise automatique	Oui, via les pairs	Oui, reprise automatique	Oui, reprise automatique	Non	Non
Sélection des relais	Dynamique, écoénergétique	Non applicable	Dynamique (pairs aléatoires)	Non applicable	Non applicable	Non applicable	Non applicable
Gestion de l'énergie verte	Oui, priorise l'énergie renouvelable	Non	Non	Non	Non	Non	Non
Priorisation des flux	Oui	Non	Non	Oui	Non	Oui	Non
Sécurité	Chiffrement et contrôle d'intégrité	Sécurité optionnelle	Variable (dépend des clients)	Sécurité optionnelle	Sécurité optionnelle	Chiffrement (TLS)	Sécurité de base (authentification)
Latence	Faible (avec relais)	Moyenne, selon les conditions	Variable (dépend du réseau)	Faible (optimisé pour réseaux à longue distance)	Faible (optimisé pour grandes distances)	Faible (optimisé pour le web)	Haute
Usage typique	Transfert de données décentralisé	Transfert massif dans les grilles	Partage de fichiers volumineux	Transferts massifs scientifiques	Transfert de fichiers distribués	Communication client-serveur (web)	Transfert de fichiers simple



Bilan du chapitre

Ce chapitre a permis de présenter en détail le processus de développement de notre protocole, ainsi que les technologies et outils utilisés pour sa réalisation. Nous avons d'abord exposé l'environnement de développement choisi, en justifiant les choix technologiques qui ont guidé la conception et la mise en œuvre du protocole. Les raisons de ces choix incluent la robustesse, la flexibilité et la performance des outils et langages sélectionnés, qui étaient essentiels pour répondre aux exigences spécifiques du protocole.

L'implémentation a ensuite été discutée, avec une attention particulière portée à la modularité du code, la gestion efficace des connexions multiplexées, et l'utilisation d'approches orientées objet pour structurer le serveur et les relais. Chaque composant a été conçu pour interagir de manière fluide, garantissant une transmission rapide et fiable des données à travers un réseau distribué.

Les résultats obtenus lors des tests de performance ont montré que notre protocole remplit ses objectifs en termes d'efficacité et de robustesse, même dans des environnements de réseau variés. Les benchmarks ont démontré que le protocole est capable de maintenir un débit élevé, de gérer efficacement les retransmissions, et de s'adapter dynamiquement aux conditions de réseau.

En conclusion, ce chapitre a permis de valider la viabilité de notre protocole dans des scénarios réels, tout en mettant en lumière les points forts de l'implémentation. Le succès des tests de performance indique que le protocole est bien positionné pour être utilisé dans des applications nécessitant des transferts de données rapides, fiables et résilients. Les résultats obtenus ouvrent également la voie à des optimisations futures et à une potentielle extension du protocole pour d'autres cas d'utilisation.



CONCLUSION GÉNÉRALE

Rappel du Problème

Dans un contexte où la rapidité, l'efficacité des transferts de données et l'optimisation énergétique sont des enjeux majeurs, en particulier dans des infrastructures utilisant un mix d'énergies renouvelables et non renouvelables, la conception d'un protocole de communication performant est essentielle. Bien que les protocoles existants soient efficaces, ils présentent des limites en termes d'efficacité énergétique et reposent principalement sur UDP et TCP. Ce travail a été entrepris pour développer un nouveau protocole, visant à améliorer à la fois la vitesse de transfert, l'efficacité énergétique en s'appuyant sur les fondations solides de QUIC.

Démarche et Résultats

Pour répondre à ces enjeux, une méthodologie rigoureuse a été adoptée. Nous avons d'abord analysé les protocoles de transfert de données existants afin d'identifier leurs forces et leurs faiblesses. Cette étude nous a permis de définir les objectifs clés pour notre protocole : une gestion optimisée des connexions multiplexées, l'intégration de relais pour répartir la charge et améliorer la résilience, l'utilisation de QUIC comme protocole de transport pour bénéficier de sa faible latence et de sa sécurité intégrée, ainsi que l'intégration d'une gestion énergétique pour privilégier les relais alimentés par des sources d'énergie renouvelables.

L'implémentation du protocole a été réalisée en utilisant le langage Go et s'est appuyée sur les implémentations QUIC de `lucas-clemente` et `MP-QUIC` de la CoNEXT 2017 [12, 13] pour garantir une compatibilité avec les environnements réseau modernes. Les résultats obtenus montrent que notre protocole surpasse les protocoles traditionnels tels que FTP dans des scénarios de transfert de fichiers volumineux, notamment en termes de débit et de gestion de la congestion. En outre, la sélection des relais en fonction de leur source d'énergie a permis de réduire significativement l'empreinte carbone des transferts de données, tout en maintenant une performance réseau optimale. Les benchmarks réalisés montrent des performances accrues grâce à l'utilisation de multiples relais et à une gestion fine des fenêtres de transfert, tout en respectant des critères énergétiques.



Limites et Perspectives

Bien que les résultats obtenus soient prometteurs, plusieurs limites subsistent. Premièrement, la complexité de la gestion des relais peut introduire une surcharge supplémentaire, notamment en termes de calcul des fenêtres de transfert et de coordination des relais. De plus, l'adaptation dynamique du protocole aux conditions réseau et énergétiques en temps réel n'a été que partiellement explorée, laissant place à des optimisations futures.

En termes de perspectives, plusieurs axes d'amélioration peuvent être envisagés. Une extension du protocole pour supporter des scénarios multi-utilisateurs avec des priorités différentes pourrait ouvrir la voie à des applications plus complexes, comme les jeux en ligne ou le streaming vidéo haute définition. Enfin, une évaluation plus poussée de la sécurité du protocole, notamment face à des attaques réseau sophistiquées, serait nécessaire pour garantir une adoption large et sécurisée.

En conclusion, ce travail a permis de poser les bases d'un protocole innovant et performant, répondant aux besoins croissants en matière de transfert de données tout en tenant compte des impératifs énergétiques. Les résultats obtenus sont encourageants, mais ils appellent à de futures recherches pour exploiter pleinement le potentiel du protocole dans des applications réelles, tout en continuant à réduire l'impact environnemental des infrastructures de communication.



RÉFÉRENCES BIBLIOGRAPHIQUES

- [1] Protocole HTTP3 sur QUIC | NetScaler 14.1. URL <https://docs.netScaler.com/fr-fr/citrix-adc/current-release/system/http3-over-quic-protocol.html>.
- [2] Structure-of-a-QUIC-packet-as-of-version-35-of-Googles-QUIC-implementation-Red-is-the.PPM (850×559). URL <https://www.researchgate.net/profile/Raman-Tenneti/publication/318914953/figure/fig13/AS:735471583899649@1552361498746/Structure-of-a-QUIC-packet-as-of-version-35-of-Googles-QUIC-implementation-Red-is-the.ppm>.
- [3] BitTorrent - Wireshark Wiki. URL <https://wiki.wireshark.org/BitTorrent>.
- [4] Saleh Alawaji. Ietf quic v1 design. *Accessed on*, 2021.
- [5] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The globus striped gridftp framework and server. In *SC '05 : Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 54–54, 2005. doi : 10.1109/SC.2005.72.
- [6] Damien Almeras. *Prenez du recul sur votre pratique grâce au modèle OSI*. OPENCLASS-ROOMS, 7 2024. <https://openclassrooms.com/fr/courses/6944606-concevez-votre-reseau-tcp-ip/7236472-prenez-du-recul-sur-votre-pratique-grace-au-modele-osi>, Visité le : 12/07/2024, à : 23 :h40.
- [7] Contributeurs aux projets Wikimedia. Internet protocol, 4 2024. URL https://fr.wikipedia.org/wiki/Internet_Protocol.
- [8] Cyrille Bousquet. Créer son protocole de jeu réseau - lecture et écriture des paquets. URL <https://bousk.developpez.com/traductions/gafferongames/construire-son-protocole-jeu-reseau/lecture-ecriture-paquets/>.
- [9] Emile Cadorel. *Energy-aware management of scientific workflows in the Cloud : a Cloud provider-centric vision*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique, 2020.
- [10] Wikipedia contributors. GridFTP, 9 2023. URL <https://en.wikipedia.org/wiki/GridFTP>.



- [11] Wikipedia contributors. BitTorrent, 9 2024. URL <https://en.wikipedia.org/wiki/BitTorrent>.
- [12] Quentin De Coninck. *Flexible multipath transport protocols*. PhD thesis, UCL-Université Catholique de Louvain, 2020.
- [13] Quentin De Coninck and Olivier Bonaventure. Multipath quic : Design and evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 160–166, 2017.
- [14] Michel Gien. A file transfer protocol (ftp). *Computer Networks (1976)*, 2(4-5) :312–319, 1978.
- [15] Pierre Giraud. *Les modèles réseaux OSI et TCP/IP expliqués - Pierre Giraud*. IONOS, 2020. <https://www.pierre-giraud.com/http-reseau-securite-cours/modele-reseau-osi-tcp-ip/>, Visité le : 17/04/2024, à : 16 :h03.
- [16] Ben Gorman. *TCP et UDP : Quelle est la différence et quel est le meilleur protocole ?* AVAST, 4 2023. <https://www.avast.com/fr-fr/c-tcp-vs-udp-difference> : :text=TCP
- [17] L'équipe Éditoriale Ionos. *Un aperçu des différents réseaux informatiques*. IONOS, 9 2019. url <https://www.ionos.fr/digitalguide/serveur/know-how/les-types-de-reseaux-informatiques-a-connaître/>, Visité le : 17/04/2024, à : 13 :h03.
- [18] L'équipe Éditoriale Ionos. *A simple and comprehensive explanation of Hypertext Transfer Protocol (HTTP)*. IONOS, 7 2020. url <https://www.ionos.com/digitalguide/hosting/technical-matters/what-is-http/>, Visité le : 17/04/2024, à : 15 :h13.
- [19] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol : Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.
- [20] Qiming Lu, Liang Zhang, Sajith Sasidharan, Wenji Wu, Phil DeMar, Chin Guok, John Macauley, Inder Monga, Se-young Yu, Jim Hao Chen, Joe Mambretti, Jin Kim, Seo-Young Noh, Xi Yang, Tom Lehman, and Gary Liu. Bigdata express : Toward schedulable, predictable, and high-performance data transfer. In *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, pages 75–84, 2018. doi : 10.1109/INDIS.2018.00011.
- [21] Constantinos Makassikis. *Modèle de coût des communications tcp à un niveau applicatif*, 2006.
- [22] Deepak Nadig, Eun-Sung Jung, Rajkumar Kettimuthu, Ian Fosterz, SV Nageswara Rao, and Byrav Ramamurthy. Comparative performance evaluation of high-performance data





transfer tools. In *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2018.

- [23] La Rédaction TechTarget. FTP (File Transfer Protocol), 8 2016. URL <https://www.lemagit.fr/definition/FTP-File-Transfer-Protocol>.
- [24] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. Multipath quic : A deployable multipath transport protocol. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7, 2018. doi : 10.1109/ICC.2018.8422951.
- [25] Liang Zhang, Wenji Wu, Phil DeMar, and Eric Pouyoul. mdtmftp and its evaluation on esnet sdn testbed. *Future Generation Computer Systems*, 79 :199–204, 2018.

