

# TRTP: Truncated Reliable Transport Protocol

## 1 Description du projet

La démocratisation de l'accès à l'espace propulse de nombreuses technologies basées sur les satellites. La société Étolien, menée par l'entrepreneuse wallonne Elwène Musc ambitionne de connecter les régions les plus reculées de Wallonie à travers son réseau de satellites de basse orbite. Grâce à des antennes cornets, véritable innovation contenant une centaine de dipôles émetteurs, déployable chez ses clients, cette entreprise offre un Internet haut-débit et de moyenne latence en tout endroit.

Pour permettre l'échange de données sur ce nouveau réseau, son département de recherche a conçu Truncated Reliable Transport Protocol (TRTP), un protocole de transport **fiable** basé sur des paquets UDP. Étolien désire une implémentation performante dans le langage **C** qui ne contient **aucune fuite de mémoire**. TRTP permettra de réaliser des transferts fiables de fichiers en utilisant la stratégie du **selective repeat**. Le selective repeat implique uniquement que le **receiver** accepte les paquets hors-séquence et les stocke dans un buffer s'ils sont dans la fenêtre de réception. Par contre, ce protocole ne permet pas au **receiver** d'indiquer au **sender** quels sont les paquets hors-séquence qu'il a reçu. TRTP permettra également la **troncation des données**<sup>1</sup>. En plus des retransmissions, le protocole propose un mécanisme de **Forward Erasure Correction (FEC)** permettant de récupérer des paquets perdus en introduisant des symboles de réparation. Comme l'architecture de ce nouveau réseau est **uniquement basée sur IPv6**, TRTP devra fonctionner au-dessus de ce protocole.

Ce projet est à faire par **groupe de deux étudiants**.

Un émetteur et un récepteur sont nécessaires pour établir un transfert de données entre deux machines distantes utilisant TRTP. Il est suggéré qu'un membre du groupe développe le **receiver** et l'autre le **sender**, tout en s'assurant de l'interopérabilité des deux programmes.

## 2 Spécifications

La spécification qui suit s'inspire des spécifications des standards de l'Internet développés par l'Internet Engineering Task Force (IETF). Cette spécification utilise les mots-clés **DOIT**, **NE PEUT PAS** et **DEVRAIT** pour spécifier les comportements du protocole. Le sens de ces mots clés est identique à celui des mots clés **MUST**, **MUST NOT** et **SHOULD** définis par l'IETF.

### 2.1 Format des segments

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

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

- (i) **PTYPE\_DATA** = 0b01 (1), indique un paquet contenant des données ;
- (ii) **PTYPE\_ACK** = 0b10 (2), indique un paquet d'acquittement de données reçues.
- (iii) **PTYPE\_NACK** = 0b11 (3), indique un paquet annonçant la réception d'un paquet de données tronquées (c.-à-d. avec le champ **TR** à 1).
- (iv) **PTYPE\_FEC** = 0b00 (0), indique un paquet FEC contenant un symbole de réparation.

**TR** Ce champ est encodé sur 1 bit. Il indique si le réseau a tronqué un payload initialement présent dans un paquet **PTYPE\_DATA**. La réception d'un paquet avec ce champ à 1 provoque l'envoi d'un paquet **PTYPE\_NACK**. Un paquet d'un type autre que **PTYPE\_DATA** avec ce champ différent de 0 **DOIT** être ignoré.

1. Tronquer les données permet au routeur de décongestionner ses buffers tout en fournissant un mécanisme de feedback rapide. Cette idée est plus détaillée dans un article récent [3].

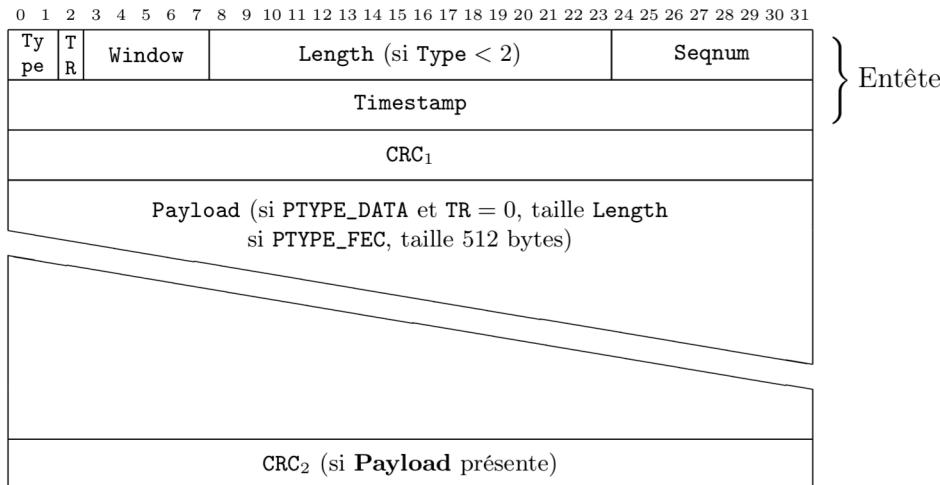


FIGURE 1 – Format des segments du protocole

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Ty pe	T R	Window		Length (si Type < 2)		Seqnum																									

FIGURE 2 – 4 premiers bytes de l’entête d’un paquet de type PTYPE\_DATA et PTYPE\_FEC

**Window** Ce champ est encodé sur 5 bits, et varie donc dans l’intervalle [0, 31]. Il indique la taille de la fenêtre de réception de l’émetteur de ce paquet, c.à.d le nombre de places vides dans le buffer de réception de l’émetteur du paquet, et peut varier au cours du temps. Si l’émetteur du paquet n’a pas de buffer de réception, ou que celui-ci est rempli, cette valeur **DOIT** être mise à 0. Le récepteur du paquet doit respecter la taille de la fenêtre annoncée, et ne pourra donc envoyer en retour des paquets de type PTYPE\_DATA dont le **Seqnum** dépasse le dernier acquittement reçu plus la taille de la fenêtre. Lors de la création d’un nouvelle connexion, l’émetteur initiant la connexion **DOIT** considérer que le destinataire a annoncé une valeur initiale de **Window** de 1.

**Length** Ce champ n’est présent que lorsque le type de paquet est PTYPE\_DATA ou PTYPE\_FEC. Les Figures 2 et 3 comparent les 4 premiers bytes de l’entête en fonction de la présence de ce champ. Ce champ est encodé sur 16 bits en network-byte order, et sa valeur est non si varie dans l’intervalle [0, 512]. Si ce champ vaut plus que 512, le paquet **DOIT** être ignoré. Dans un paquet PTYPE\_DATA, il dénote le nombre d’octets de données dans le champ **Payload**. Un paquet PTYPE\_DATA avec ce champ à 0 marque le numéro de séquence précédent comme étant le dernier du transfert. Une éventuelle troncation du paquet ne change pas la valeur de ce champ. Lorsque ce champ est absent, sa valeur est considérée comme étant 0. Le champ **Length** d’un paquet PTYPE\_FEC contient un symbole de réparation tel que décrit dans la section 2.2.

**Seqnum** Ce champ est encodé sur 8 bits, et sa valeur varie dans l’intervalle [0, 255]. Sa signification dépend du type du paquet.

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

**PTYPE\_ACK** Il correspond au numéro de séquence du prochain numéro de séquence attendu (c-à-d (le dernier numéro de séquence + 1) % 2<sup>8</sup>). Il est donc possible d’envoyer un seul paquet PTYPE\_ACK qui sert d’acquittement pour plusieurs paquets PTYPE\_DATA (principe des acquis cumulatifs) ;

**PTYPE\_NACK** Il correspond au numéro de séquence du paquet tronqué qui a été reçu. S’il ne rentre pas dans la fenêtre des numéros de séquence envoyés par l’émetteur, celui-ci **DOIT** ignorer le paquet.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type	T R	Window	Sqnum	Timestamp	...																										

FIGURE 3 – 4 premiers bytes de l’entête d’un paquet de type PTYPE\_ACK et PTYPE\_NACK

**PType\_FEC** Il correspond au numéro de séquence de ce symbole de réparation FEC. Un numéro  $n$  indique que ce symbole a été produit à partir des paquets  $n$ ,  $n + 1$ ,  $n + 2$  et  $n + 3$ , tel que décrit dans la section 2.2.

Lorsque l’émetteur atteint le numéro de séquence 255, le prochain numéro de séquence est 0 ;

**Timestamp** Ce champ est encodé sur 4 octets, et représente une valeur dont la signification est laissée libre aux implémentateurs. Il n’y a donc pas de considération particulière sur son endianness. Pour chaque paquet PTYPE\_DATA et PTYPE\_FEC, son émetteur choisit une valeur pour ce champ. Lorsque le destinataire envoie un paquet PTYPE\_ACK ou PTYPE\_NACK, il indique dans ce champ la valeur du champ **Timestamp** du dernier paquet PTYPE\_DATA reçu ou PTYPE\_FEC utilisé.

**CRC1** Ce champ est encodé sur 4 octets, en network byte-order. Ce champ contient le résultat de l’application de la fonction CRC32<sup>2</sup> à l’entête avec le champ **TR** mis à 0<sup>3</sup>, juste avant qu’il ne soit envoyé sur le réseau. À la réception d’un paquet, cette fonction doit être recalculée sur l’entête avec le champ **TR** mis à 0, et le paquet **DOIT** être ignoré si les deux valeurs diffèrent.

**Payload** Ce champ contient au maximum 512 octets. Dans un paquet PTYPE\_DATA, si le champ **TR** est 0, sa taille est donnée par le champ **Length** ; sinon, sa taille est nulle. Il contient les données transportées par le protocole. Dans un paquet PTYPE\_FEC, sa taille est de 512 bytes et il contient un symbole de réparation tel que décrit dans la section 2.2.

**CRC2** Ce champ est encodé sur 4 octets, en network byte-order. Ce champ contient le résultat de l’application de la fonction CRC32 à l’éventuel champ **Payload**, juste avant qu’il ne soit envoyé sur le réseau. Ce champ n’est présent que si le paquet contient un champ **Payload** et qu’il n’a pas été tronqué (champ **TR** à 0). À la réception d’un paquet avec ce champ, cette fonction doit être recalculée sur le payload, et le paquet **DOIT** être ignoré si les deux valeurs diffèrent.

[Les 8 premières tâches du cours INGInious](#) vous permettent d’exercer votre compréhension du format du protocole.

Les objets communicants à travers ce réseau peuvent être répartis à travers le monde. Le consortium veut donc que TRTP fonctionne correctement dans le modèle de réseau suivant :

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

## 2.2 Utilisation de FEC

Le protocole TRTP permet de protéger un transfert par l’envoi de paquets PTYPE\_FEC. Ces paquets contiennent des symboles de réparation utilisés pour récupérer des données manquantes à partir de données reçues. Cette section décrit plus en détails l’utilisation de ces paquets.

Comme illustré par la Figure 4, un paquet PTYPE\_FEC de séquence  $n$  peut-être généré à partir de quatre paquets PTYPE\_DATA de séquence  $n$ ,  $n + 1$ ,  $n + 2$  et  $n + 3$ . La particularité du paquet PTYPE\_FEC est la façon dont sont générées les valeurs des champs **Length** et **Payload**. Chaque champ du paquet

2. L’implémentation la plus courante de cette fonction se trouve dans **zlib.h**.
3. Un routeur tronquera les paquets lorsque ses charges réseau ou CPU deviennent importantes. Le champ **TR** n’est pas protégé par le CRC pour que le routeur puisse changer sa valeur sans devoir recalculer le CRC.

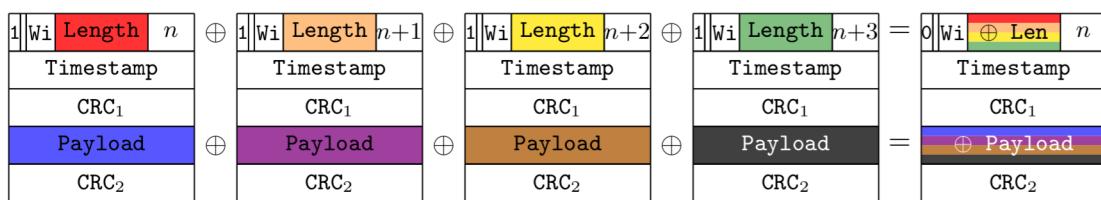


FIGURE 4 – Génération d'un paquet PTYPE\_FEC à partir de quatre paquets PTYPE\_DATA.

PTYPE\_FEC correspond à l'application de l'opération XOR sur le champ correspondant des quatre paquets PTYPE\_DATA.

L'intérêt du paquet PTYPE\_FEC est que cette opération XOR est réversible. On peut donc reconstituer un des paquets PTYPE\_DATA avec le paquet PTYPE\_FEC correspondant et 3 des 4 paquets PTYPE\_DATA.

Un paquet PTYPE\_FEC n'est pas acquitté. S'il est utilisé pour récupérer des données manquantes, le numéro de séquence correspondant au paquet récupéré transportant ces données manquantes **DOIT** être acquitté. Un paquet PTYPE\_FEC qui ne peut être directement utilisé **DOIT** être ignoré. Quand le mécanisme de FEC est activé pour un transfert, l'émetteur **DEVRAIT** envoyer un paquet PTYPE\_FEC pour 4 paquets PTYPE\_DATA.

### 2.2.1 Opération XOR sur un tableau de bytes

Le XOR ( $\oplus$ ) entre deux tableau de bytes se calcule en appliquant l'opération XOR byte comme suit :

$$(a \oplus b)[i] = a[i] \oplus b[i]$$

Lorsque les tableaux ne sont pas de taille égale, des zéros sont ajoutés à la fin du plus petit tableau pour atteindre la même taille.

## 2.3 Programmes

L'implémentation du protocole devra permettre de réaliser un transfert de données unidirectionnel au moyen de deux programmes, **sender** et **receiver**. Vous serez chargés de l'implémentation de ces deux programmes. Ces programmes devront être produits au moyen de la cible par défaut d'un Makefile. Votre implémentation **DOIT** fonctionner sur les ordinateurs de la salle Intel, bâtiment Réaumur<sup>4</sup>.

Chaque programme nécessite deux arguments pour se lancer : **hostname** et **port**. **hostname** est un nom de domaine ou une adresse IPv6 et **port** est un numéro de port UDP. Pour le programme **sender**, ils désignent le **receiver** à contacter. Pour le programme **receiver**, ils désignent la destination de la connexion à accepter<sup>5</sup>.

**Important :** Il est fortement recommandé d'utiliser l'appel système **poll** et non des threads ou processus supplémentaires pour la réalisation du projet.

Sans autres arguments, le **sender** lit sur l'entrée standard et envoie son contenu au **receiver**. Ce contenu se termine au signalement d'EOF. Sans autres arguments également, le **receiver** attend la connexion entrante d'un **sender**. Le fichier reçu est imprimé sur la sortie standard. Le **receiver** s'arrête après avoir reçu l'entièreté du fichier transféré par le **sender**.

Par ailleurs, le programme **sender** supportera deux arguments optionnels :

**-f filename** : spécifie un fichier nommé **filename** à envoyer. Seule la présence du fichier est garantie, son contenu peut être de différents types et donc **pas nécessairement du texte**.

**-c** : active l'envoi de paquets PTYPE\_FEC durant le transfert.

4. [Les machines peuvent être accédées via SSH.](#)

5. :: est l'adresse désignant toutes les destinations réseaux.

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

Vos programmes devront également fournir un ensemble de statistiques après l'envoi ou la réception d'un fichier. Ces indicateurs sont aussi utile lors de l'implémentation du protocole, il est donc conseillé de les ajouter au fur et à mesure de votre avancement. Ces statistiques sont imprimées sur la sortie d'erreur, ou vers un fichier renseigné par l'argument optionnel **-s filename**. Elles prennent la forme d'un fichier CSV dont le format et le contenu sont décrits dans l'annexe A.

Voici quelques exemples pour lancer les programmes :

<code>sender ::1 12345 &lt; fichier.dat</code>	Redirige le contenu du fichier <code>fichier.dat</code> sur l'entrée standard, et l'envoie sur le <code>receiver</code> présent sur la même machine ( <code>::1</code> ), qui écoute sur le port 12345.
<code>sender -f fichier.dat ::1 12345</code>	Idem, mais sans passer par l'entrée standard.
<code>sender -f fichier.dat localhost 12345</code>	Idem, en utilisant un nom de domaine (localhost est défini dans <code>/etc/hosts</code> ).
<code>receiver :: 12345 2&gt; log.txt</code>	Lance un <code>receiver</code> qui écoute sur toutes les interfaces. Le fichier reçu est imprimé sur la sortie standard. Les messages d'erreur et de log sont redirigés de la sortie d'erreur vers <code>log.txt</code> .
<code>receiver :: 12345 -s stats.csv 2&gt; log.txt</code>	Idem, mais les statistiques sont écrites dans le fichier <code>stats.csv</code> au lieu de se mélanger aux messages d'erreur et logs dans <code>log.txt</code> .

## 2.4 Étapes conseillées pour réaliser le projet

Pour vous faciliter la réalisation du projet, nous vous proposons cette liste de sous-tâches que vous devriez effectuer.

1. Implémentation de l'encodage et du décodage des paquets (en particulier, faites attention à l'endianness des champs) dans une approche orientée tests. **Une tâche INGInious est prévue pour vous aider**, il est fortement conseillé de la réaliser.
2. Implémentation d'un `sender` et d'un `receiver` simple sur UDP via l'API `socket`.
3. Fonctionnement du protocole TRTP avec un échange d'un paquet unique sur un réseau parfait.
4. Fonctionnement du protocole TRTP avec des échanges de n'importe quelle taille avec le respect de toutes les spécifications sur un lien qui présente juste de la latence (par exemple, 200 ms), avec une suite de tests black-box vérifiant cela.
5. Fonctionnement du protocole TRTP sous toutes les conditions réseaux possibles (pertes, corruption, troncation, jitter, latence) avec une suite de tests black-box vérifiant cela.
6. Implémentation de l'utilisation des paquets `PTYPE_FEC`.
7. Évaluation des performances et identification des sources de limitations de votre implémentation.

## 3 Tests

### 3.1 INGInious

Des tâches INGInious sont fournies pour vous familiariser avec deux facettes du projet :

1. **L'envoi et la réception de donnée sur le réseau, multiplexés sur un socket.**
2. **La création, l'encodage et le décodage de paquets**

Elles ne seront pas évalués directement pour le projet mais vous permettent de vous exercer en langage C et de préparer des morceaux de fonctionnalités pour le projet.

### 3.2 Tests individuels

Les tests INGInious ne seront pas suffisant pour tester votre implémentation. Il vous est donc demandé de tester par vous-même votre code afin de réaliser une suite de test et de la documenter dans votre rapport.

### 3.3 Tests d'interopérabilité

Vos programmes **doivent** être inter-opérable avec les implémentations d'autres groupes. Vous ne pouvez donc pas créer de nouveau type de paquets, ou rajouter des méta-données dans les champs. Une semaine avant la remise de la deuxième soumission, vous **devrez** tester votre implémentation avec 2 autres groupes (votre **sender** et leurs **receiver** et votre **receiver** et leurs **sender**).

## 4 Planning et livrables

### A. Première soumission, 23/03 à 19h, sur INGInious

1. Implémentation du **receiver** et **sender** suivant les spécifications du protocole.
2. Suite de tests des programmes.
3. Makefile dont la **cible par défaut** produit les deux programmes dans le répertoire courant avec comme nom **sender** et **receiver**, et dont la cible **tests** lance votre suite de tests.

### B. Tests d'inter-opérabilité, durant les séances de TP du 24/03 au 31/03 inclus (plages horaires et modalités à confirmer)

### C. Soumission finale, 27/04 à 19h, sur INGInious

Mêmes critères que pour la première soumission, avec un rapport (max 4 pages, en PDF), décrivant l'architecture générale de votre projet, et répondant **au minimum** aux questions suivantes :<sup>6</sup>

- Comment avez-vous implémenté le mécanisme de fenêtre de réception ?
- Quel est votre stratégie pour la génération des acquittements ?
- Comment gérez-vous la fermeture de la connexion ?
- Que mettez-vous dans le champ Timestamp, et quelle utilisation en faites-vous ?
- Comment avez-vous choisi la valeur du timer de retransmission ?
- Comment réagissez-vous à la réception de paquets PTYPE\_NACK ?
- Si le **receiver** ne peut traiter votre ouverture de connexion (p. ex. il est surchargé ou injoignable), quelle est votre stratégie ?
- Quelle est la partie critique de votre implémentation, affectant la vitesse de transfert ? (**Justifiez**)
- Quelle est votre stratégie d'envoi des paquets PTYPE\_FEC ?
- Quel impact à la gestion des paquets PTYPE\_FEC sur la fenêtre de réception du **receiver** ?
- Quelles sont les performances de votre implémentation ? (**évaluation** à l'aide de graphes et de chiffres, scénarios explorés, explication de la méthodologie)
- Quelle(s) stratégie(s) de tests avez-vous utilisée(s) ? (**Justifiez**)

De plus le rapport devra décrire en plus le résultat des tests d'interopérabilité en annexe, ainsi que les changements effectués au code si applicable.

#### Format des livrables

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

---

6. Le rapport sera lu par les experts d'Étoilien qui connaissent le sujet. Soyez donc **objectifs** et évitez de réintroduire le contexte du projet dans votre rapport. Toutefois, libre à vous de mettre en avant une partie du projet non listée dans la liste ci-dessous que vous trouvez pertinente.

/	Racine de l'archive
- Makefile	Le Makefile demandé
- src/	Le répertoire contenant le code source du <b>receiver</b> et <b>sender</b>
- tests/	Le répertoire contenant la suite de tests
- rapport.pdf	Le rapport
- gitlog.stat	La sortie de la commande <code>git log --stat</code>

Cette archive sera nommée `projet1_nom1_nom2.zip`, où nom1/2 sont les noms de famille des membres du groupe, et sera à soumettre sur deux tâches dédiées sur INGInious (une pour chaque soumission) qui vérifieront le format de votre archive. Si celle-ci ne passe pas les tests de format, **votre soumission ne sera pas considérée pour l'évaluation !** Les liens de ces tâches vous seront communiqués en temps utile.

Vous **devez** réaliser le projet en utilisant le gestionnaire de version `git`.

 **Important :** l'évaluation tiendra compte de vos deux soumissions, et pénalisera les projets dont la première soumission suggère que le travail a été bâclé. La première soumission n'est donc **PAS** facultative.

## 5 Évaluation

La note du projet sera composée des trois parties suivantes :

1. Implémentation : Vos programmes fonctionnent-ils correctement ? Sont-ils interopérables ? Qualité de la suite de tests ? Que se passe-t-il quand le réseau est non-idéal ? ...  
Le respect des spécifications (arguments, fonctionnement du protocole, formats des paquets, structure de l'archive, ...) est impératif à la réussite de cette partie !
2. Votre rapport.
3. Peer-review individuelle du code de deux autres groupes. Elle sera effectuée durant la semaine suivant la remise du projet. Vous serez noté sur la pertinence de vos commentaires. Plus d'informations suivront.

## 6 Ressources utiles

Les manpages des fonctions suivantes sont un bon point de départ pour implémenter le protocole, ainsi que pour trouver d'autres fonctions utiles : `socket`, `bind`, `getaddrinfo`, `connect`, `send`, `sendto`, `sendmsg`, `recv`, `recvfrom`, `recvmsg`, `select`, `poll`, `getsockopt`, `read`, `write`, `fcntl`, `getnameinfo`, `htonl`, `ntohl`, `getopt`

[2] est un tutoriel disponible en ligne, présentant la plupart des appels systèmes utilisés lorsque l'on programme en C des applications interagissant avec le réseau. **C'est probablement la ressource qui vous sera le plus utile pour commencer votre projet.**

[5] et [4] sont deux livres de références sur la programmation système dans un environnement UNIX, disponibles en bibliothèque INGI.

[1] est le livre de référence sur les sockets TCP/IP en C, disponible en bibliothèque INGI.

## Références

- [1] Michael J. Donahoo and Kenneth L. Calvert. TCP / IP sockets in C, A practical guide for programmers, 2001.
- [2] Brian "Beej Jorgensen" Hall. Beej's guide to network programming, 2018. URL : <https://beej.us/guide/bgnet/html/multi/index.html>.

- [3] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 29–42. ACM, 2017, available at <http://conferences2.sigcomm.org/sigcomm/2017/program.html>.
- [4] Michael Kerrisk. The Linux programming interface : a Linux and UNIX system programming handbook, 2010.
- [5] W. Richard Stevens and Stephen A. Rago. Advanced programming in the Unix environment, 2005.

## A Format et contenu des statistiques de transferts

Le format des statistiques est un fichier CSV à deux colonnes, où les valeurs sont séparées par le caractère ":". La première colonne indique le nom de la statistique, la deuxième sa valeur entière. Les statistiques suivantes **DOIVENT** faire partie de cet ensemble.

<b>data_sent</b>	Nombre de paquet de type PTYPER_DATA envoyés.
<b>data_received</b>	Nombre de paquet de type PTYPER_DATA valides reçus.
<b>data_truncated_received</b>	Nombre de paquet de type PTYPER_DATA valides reçus avec le champ <b>TR</b> à 1.
<b>fec_sent</b>	Nombre de paquet de type PTYPER_FEC envoyés.
<b>fec_received</b>	Nombre de paquet de type PTYPER_FEC valides reçus.
<b>ack_sent</b>	Nombre de paquet de type PTYPER_ACK envoyés.
<b>ack_received</b>	Nombre de paquet de type PTYPER_ACK valides reçus.
<b>nack_sent</b>	Nombre de paquet de type PTYPER_ACK envoyés.
<b>nack_received</b>	Nombre de paquet de type PTYPER_ACK valides reçus.
<b>packet_ignored</b>	Nombre de paquet ignorés.
<b>sender</b> uniquement	
<b>min_rtt</b>	Temps minimum en millisecondes entre l'envoie d'un paquet PTYPER_DATA et la réception de l'acquittement correspondant.
<b>max_rtt</b>	Temps maximum en millisecondes entre l'envoie d'un paquet PTYPER_DATA et la réception de l'acquittement correspondant.
<b>packet_retransmitted</b>	Nombre de paquets PTYPER_DATA retransmis à la suite d'une perte, troncation ou corruption.
<b>receiver</b> uniquement	
<b>packet_duplicated</b>	Nombre de paquets PTYPER_DATA reçus étant déjà dans le buffer de réception.
<b>packet_recovered</b>	Nombre de paquets PTYPER_DATA récupérés grâce aux paquets PTYPER_FEC.

FIGURE 5 – Définition des statistiques de transferts

Vous êtes libres d'étendre cet ensemble avec d'autres statistiques. Vous devez alors expliquer leur signification dans le rapport.

La Figure 6 illustre un exemple de statistiques fournies par le **sender**.

```
data_sent:12
data_received:0
fec_sent: 3
fec_received: 0
data_truncated_received:0
ack_sent:0
ack_received:9
nack_sent:0
nack_received:1
packets_ignored:0
min_rtt:18
max_rtt:29
packets_retransmitted:1
```

FIGURE 6 – Exemple de statistiques fournies par le **sender**