TRTP: Truncated Reliable Transport Protocol

1 Description du projet

Une nouvelle entreprise de l'Internet des Objets, *Intelligent Data for the Internet Of Things*, a récemment installé un nouveau data-center dans la région. Celui-ci lui permettra l'acheminement et la collecte des données issus des senseurs de ses clients. Cette entreprise valorise alors ces données en effectuant de la prédiction statistique intensive. Ces données sont donc critiques pour sa réussite et nécessitent un protocole de transport pour leur acheminement jusqu'au data-center.

Pour se faire, elle a conçu Truncated Reliable Transport Protocol (TRTP), un protocole de transport **fiable** basé sur des segments **UDP**. Elle désire une implémentation performante dans le langage **C** qui ne contient **aucune fuite de mémoire** [1]. TRTP permettra de réaliser des transferts fiables de fichiers, utilisera la stratégie du **selective repeat** [2] et permettra la **troncation des données** [3]. Comme l'architecture du réseau de l'entreprise 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 recepteur sont nécessaires pour établir un transfert de données entre deux machines distantes utilisant TRTP. Votre groupe sera désigné responsable de la réalisation **d'un seul des deux programmes**.

2 Spécifications

2.1 Protocole

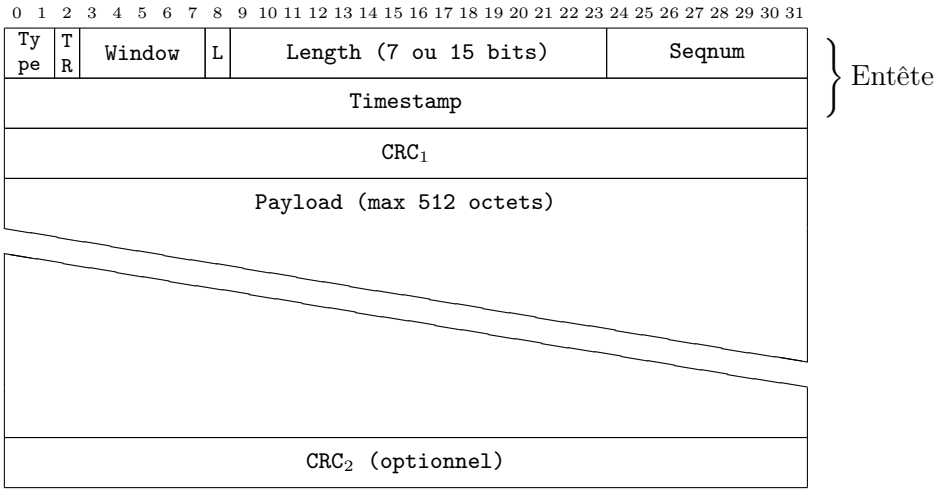

FIGURE 1 – Format des paquets du protocole

Le format des segments du protocole est visible sur la Figure 3. Ils se composent des champs dans l'ordre suivant :

1. Celles-ci peuvent faire crasher des machines de collectes ou les senseurs, impliquant un coût de maintenance et une perte de données tout deux conséquents.
2. 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-ci ne permet pas au `receiver` d'indiquer au `sender` quels sont les paquets hors-séquence qu'il a reçu.
3. Tronquer les données peut permettre au réseau de décongestionner ses buffers tout en fournissant un mécanisme de feedback rapide. Cette idée est plus détaillée dans un article récent [3].

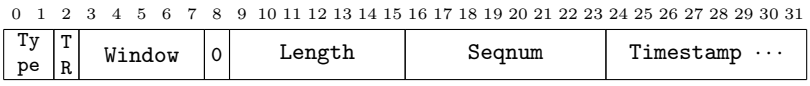

FIGURE 2 – 4 premiers bytes de l'entête lors de l'utilisation du champ Length si `L` = 0

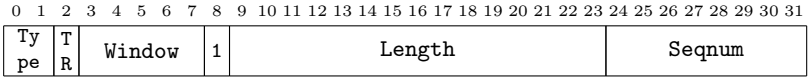

FIGURE 3 – 4 premiers bytes de l'entête lors de l'utilisation du champ Length si `L` = 1

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

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

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

(iii) `PTYPE_NACK` = 3, indique un paquet annonçant la réception d'un paquet de données tronquées (i.e. avec le champ TR à 1).

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

TR Ce champ est encodé sur 1 bit, et 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 ;

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. Cette valeur indique le nombre de places vides dans le buffer de réception de l'émetteur du segment, et peut varier au cours du temps. Si un émetteur n'a pas de buffer de réception, cette valeur **DOIT** être mise à 0. Un émetteur ne peut envoyer de nouvelles données avant d'avoir reçu un acquit pour le paquet précédant que si le champ **Window** du dernier paquet provenant du destinataire était non nul. Lors de la création d'un nouvelle connexion, l'émetteur initiant la connexion **DOIT** considérer que le destinataire avait annoncé une valeur initiale de **Window** valant 1.

L Ce champ est encodé sur 1 bit, et indique la longueur du champ **Length** qui suit. Une valeur de 0 indique que le champ **Length** est encodé sur 7 bits, tandis qu'une valeur de 1 indique qu'il est encodé sur 15 bits.

Length Ce champ est encodé sur 7 ou 15 bits en network-byte order, et sa valeur varie dans l'intervalle [0, 512]. Sa taille est déterminée sur base du drapeau **L** précédent. Il dénote le nombre d'octets de données dans le champ **Payload**. Un paquet `PTYPE_DATA` avec ce champ à 0 et dont le numéro de séquence correspond au dernier numéro d'acquittement envoyé par le destinataire signifie que le transfert est fini. Une éventuelle troncation du paquet ne change pas la valeur de ce champ. Si ce champ vaut plus que 512, le paquet **DOIT** être ignoré.

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 segment d'une connexion a le numéro de séquence 0. Si le numéro de séquence ne rentre pas dans la fenêtre des numéros de séquence autorisés par le destinataire, celui-ci **DOIT** ignorer le paquet ;

`PTYPE_ACK` Il correspond au numéro de séquence du prochain numéro de séquence attendu (c-à-d (le dernier numéro de séquence + 1) $\%2^8$). 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.

Lorsque l'émetteur atteint le numéro de séquence 255, il recommence à 0 ;

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

CRC1 Ce champ est encodé sur 4 octets, en network byte-order. Ce champ contient le résultat de l'application de la fonction CRC32 [4] à l'entête avec le champ TR mis à 0 [5], 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. Il contient les données transportées par le protocole. Si le champ TR est 0, sa taille est donnée par le champ **Length** ; sinon, sa taille est nulle.

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 senseurs communicant avec le data-center de l'entreprise peuvent être répartis à travers le monde. L'entreprise veut donc que TRTP fonctionne correctement dans le modèle de réseau suivant :

1. Un segment de données envoyé par un hôte est reçu **au plus une fois** (pertes mais pas de duplication) ;
2. Le réseau peut **corrompre** les segments de données de façon aléatoire ;
3. Le réseau peut **tronquer** le payload des paquets de façon aléatoire ;
4. 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 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 d'**un seul** 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 **Linux** de la salle Intel, bâtiment Réaumur [6].

Chaque programme nécessite au minimum deux arguments pour se lancer : `hostname` et `port`. `hostname` est un nom de domaine ou une adresse IPv6 et `port` est le numéro de port UDP. Pour le programme `sender`, ils permettent de désigner le `receiver` à contacter. Pour le programme `receiver`, ils désignent la source des connexions à accepter [7].

☞ **Important :** Il est fortement recommandé d'utiliser l'appel `select` et non des threads ou processus supplémentaires pour la réalisation de votre programme.

4. Le diviseur (polynomial) de cette fonction est $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$, la représentation normale de ce polynôme (en network byte-order) est $0\times04C11DB7$. L'implémentation la plus courante de cette fonction se trouve dans `zlib.h`, mais de nombreuses autres existent.

5. En règle générale, un routeur commencera à tronquer les paquets lorsque ses charges réseau et/ou CPU deviendront importantes. On voudrait donc éviter que ce dernier ait à recalculer le CRC (et donc perdre du temps CPU) si le champ **TR** change de valeur.

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

7. `::` est l'adresse désignant toutes les sources réseaux.

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 des connexions entrantes de plusieurs `sender`[8] et stocke chaque fichier reçu. Les fichiers reçus sont stockés dans le répertoire courant et sont numérotés à partir de zero dans l'ordre dans lequel les connexions desquelles ils sont issus sont ouvertes.

Par ailleurs, le programme `sender` supportera l'argument optionnel `-f filename`, qui spécifie un fichier nommé `filename` à envoyer. Seul la présence du fichier est garantie, son contenu peut être de différents types et donc **potentiellement binaire**. Le programme `receiver` supportera l'argument optionnel `-o format`, qui spécifie un format de formattage pour les fichiers reçus. Ce format est conforme avec la spécification du format des fonctions `printf`, `sprintf` et autres variantes[9]. Il **DOIT** spécifier un seul entier à convertir, e.g. `%d` ou `fichier_%02d.dat`. L'argument `-m N` spécifie le nombre N de connexions que le `receiver` **DOIT** pouvoir traiter de façon concurrente.

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

Voici quelques exemples pour lancer les programmes :

`sender ::1 12345 < fichier.dat`	Redirige le contenu du fichier fichier.dat sur l'entrée standard, et l'envoie sur le `receiver` présent sur la même machine (`::1`), qui écoute sur le port 12345.
`sender -f fichier.dat ::1 12345`	Idem, mais sans passer par l'entrée standard.
`sender -f fichier.dat localhost 12345`	Idem, en utilisant un nom de domaine (localhost est défini dans /etc/hosts).
`receiver -m 4 :: 12345 2> log.txt`	Lance un receiver qui écoute sur toutes les interfaces. Il traitera au maximum 4 connexions concurrentes. Chaque fichier reçu est stocké dans le répertoire courant. Les messages d'erreur et de log sont redirigés dans log.txt.
`receiver -o "fichier_%02d.dat" :: 12345`	Lance un receiver qui écoute sur toutes les interfaces. Chaque fichier reçu est stocké dans le répertoire courant et suivent les noms `fichier_00.dat`, `fichier_01.dat`, ... Les messages d'erreur et de log sont imprimés sur la sortie d'erreur standard.

2.3 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[10] est prévue pour vous aider, il est fortement conseillé de la réaliser.
2. Implémentation d'un `sender` ou 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. (Pour le `receiver` uniquement) Implémentation des connexions simultanées.
5. 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.
6. Fonctionnement du protocole TRTP sous toutes les conditions réseaux possibles (pertes, corruption, troncation, jitter, latence) avec une suite de tests black-box vérifiant cela.
7. Évaluation des performances et identification de la partie critique de votre implémentation.

8. Jusqu'à 100 par défaut.
9. `http://man7.org/linux/man-pages/man3/printf.3.html`
10. `https://inginious.info.ucl.ac.be/course/LINGI1341/format-des-segments`

3 Tests

3.1 INGInious

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

1. La création, l'encodage et le décodage de segments [11] ;
2. L'envoi et la réception de donnée sur le réseau, multiplexés sur un socket [12].

Elles ne sont pas en lien direct avec le projet mais vous permettent de vous exercer en langage C.

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 le documenter dans votre rapport.

3.3 Tests d'interopérabilité

Votre programme **DOIT** être inter-opérable avec les implémentations d'autres groupes. Ne vous occupant que d'un seul des deux programmes requis, l'absence d'interoperabilité le rend inutile. Vous ne pouvez donc pas créer de nouveau type de segments, ou rajouter des méta-données dans le payload. Deux possibilités de tester l'interopérabilité de votre implémentation s'offrent à vous.

1. Vous aurez la possibilité de tester votre programme avec un autre programme de référence. E.g. votre `sender` avec un `receiver` de référence ou l'inverse.
2. Une semaine avant la remise de la deuxième soumission, vous **devrez** tester votre implémentation avec 2 autres groupes (votre `sender` et leurs `receiver` ou votre `receiver` et leurs `sender`).

4 Planning et livrables

A. Première soumission, 22/10 à 18h, sur INGInious

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

B. Tests d'inter-opérabilité, durant les séances de TP de la semaine du 24/10 au 29/10 inclus, en salle Intel (plages horaires à confirmer)

C. Soumission finale, 31/10 à 18h, sur INGInious

Même critères que pour la première soumission, avec un rapport (max 4 pages, en PDF), décrivant l'architecture générale de votre programme, et répondant **au minimum** aux questions suivantes : [13]

Si vous êtes responsable de la réalisation du `sender` :

— Que mettez-vous dans le champ Timestamp, et quelle utilisation en faites-vous ?

— Comment réagissez-vous à la réception de paquets `PTYPE_NACK`

— Comment avez-vous choisi la valeur du timer de retransmission ?

— Si le `receiver` ne peut traiter votre ouverture de connexion (e.g. il est surchargé ou injoignable), quelle est votre stratégie ?

11. `https://inginious.info.ucl.ac.be/course/LINGI1341/encoder-decoder-structure`
12. `https://inginious.info.ucl.ac.be/course/LINGI1341/envoyer-et-recevoir-des-donnees`
13. Le rapport sera lu par des experts mandatés par l'entreprise qui connaissent le sujet. Soyez donc **objectifs** et évitez de réintroduire le contexte du projet dans votre rapport. Toutefois, libre à vous de mettre en avant une partie du projet non listée dans la liste ci-dessous que vous trouvez pertinente.

Si vous êtes responsable de la réalisation du `receiver` :

— Comment maintenez-vous l'état des différents connections concurrentes ?

— Quelle est votre technique pour traiter les connections concurrentes ?

— Comment avez-vous implémenté le mécanisme de fenêtre de réception ?

— Quel est votre stratégie pour la génération des acquittements ?

Enfin, voici les questions communes aux deux programmes :

— Quelle est la partie critique de votre implémentation, affectant la vitesse de transfert ?

— Comment gérer vous la fermeture de la connexion ?

— 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) ?

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 :

`/`	Racine de l'archive
`|- Makefile`	Le Makefile demandé
`|- src/`	Le répertoire contenant le code source du programme désigné
`|- tests/`	Le répertoire contenant la suite de tests
`|- rapport.pdf`	Le rapport
`|- gitlog.stat`	La sortie de la commande `git log --stat`

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

Il est demandé de réaliser le projet en utilisant le gestionnaire de version `git`.

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

5 Evaluation

La note du projet sera composée des trois parties suivantes :

1. Implémentation : Votre programme fonctionne t'il correctement ? Est-t-il interopérable ? 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 segments, 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.**

[6] 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.

[5] et `man select_tut` donnent une introduction à l'appel système `select`, utile pour lire et écrire des données de façon non-bloquante.

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] Spencer Low. The world of select(). URL : `http://www.lowtek.com/sockets/select.html`.

[6] W. Richard Stevens and Stephen A. Rago. Advanced programming in the Unix environment, 2005.