Implémentation d'un protocole de transport sans pertes

1 Description du projet

Ce projet est à faire par **groupe de deux**. Le but du projet est d'implémenter en language **C** un protocole de transport utilisant des segments **UDP**. Ce protocole permettra de réaliser des transferts fiables de fichiers, et utilisera la stratégie du **selective repeat**. Il devra également fonctionner avec **IPv6**. En outre, ce protocole supportera une variante de l'approche présentée dans l'article [Che+14]. Ainsi, il permettra à la destination d'envoyer un message à la source du transfert afin de lui signaler la non-réception des données d'un segment en cas de congestion. Deux programmes sont à réaliser, permettant de faire un transfert de données entre deux machines distantes.

2 Spécifications

2.1 Protocole

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

Type Ce champ est encodé sur 3 bits. Il indique le type du paquet, trois types 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` = 4, indique qu'un paquet de données est manquant. Un paquet avec un autre type DOIT être ignoré ;

Window Ce champ est encodé sur 5 bits, et varie 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, et peut donc par définition varier au cours du temps. Si l'émetteur n'a pas de buffer de réception, cette valeur DEVRAIT être mise à 0. Un émetteur ne peut envoyer de nouvelles données avant d'avoir reçu un acquit pour le paquet précédant que s'il y a assez de place dans le buffer du destinataire.

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 connection 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 inuméro de séquence du prochain numéro de séquence attendu (c-à-d (le dernier numéro de séquence + 1) mod 2^8). Il est donc possible d'envoyer un seul paquet `PTYPE_ACK` qui sert d'acquittement pour plusieurs paquets `PTYPE_DATA` ;

`PTYPE_NACK` Il correspond au numéro de séquence d'un paquet qui a été coupé dans le réseau avant d'atteindre son destinataire, et est destiné à l'émetteur afin de lui notifier qu'il y a eu de la congestion dans le réseau.

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

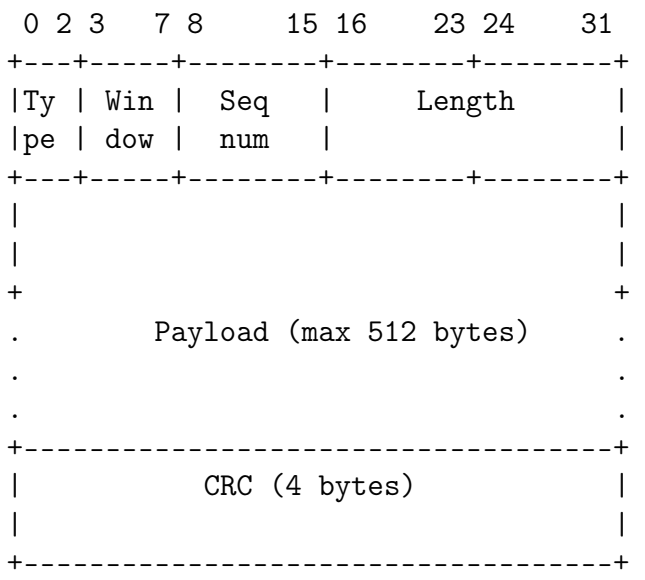

FIGURE 1 – Format des paquets du protocole

Length Ce champ est encodé sur 16 bits, et sa valeur varie dans l'intervalle [0, 512]. Il dénote le nombre de bytes de données utiles dans le payload (et donc exclut un éventuel padding). 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. Si ce champ vaut plus que 512, le paquet DOIT être ignoré ;

Payload Ce champ contient au maximum 512 bytes, et préserve l'alignement 32 bits du paquet. Il contient les données transportées par le protocole. Si le nombre de bytes des données n'est pas un multiple de 4, un padding avec des bytes mis à 0 doit être présent pour préserver l'alignement. Le champ Length permettra alors de le signaler car il ne sera plus un multiple de 4 ;

CRC Ce champ contient le résultat de l'application de la fonction CRC32 [1] au segment (header + éventuel payload), juste avant qu'il ne soit envoyé sur le réseau. À la réception d'un paquet, cette fonction doit être recalculée sur celui-ci, et le paquet DOIT être ignoré si les deux valeurs diffèrent.

Le modèle du réseau dans lequel ce protocole fonctionne est le 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. En cas de **congestion** dans le réseau, les switchs/routeurs adoptent le comportement suivant en fonction du type du segment :

   `PTYPE_DATA` Le switch/routeur coupe le paquet et **ne conserve que le header du segment** (les 4 premiers bytes du paquet). Celui-ci est alors transmis tel quel à la destination (il y a donc une incohérence entre la longueur du paquet reçu, la valeur du champ Length, et le fait que le packet ne contient plus le champ CRC). Le destinataire DEVRAIT réagir en envoyant un segment `PTYPE_NACK` à la source, avec le numéro de séquence du header reçu ;

   `PTYPE_ACK` Le paquet est **effacé** ;

1. 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 $0x04C11DB7$. L'implémentation la plus courante de cette fonction se trouve dans `zlib.h`, mais de nombreuses autres existent.

`PTYPE_NACK` Le paquet est**effacé**.

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 unidirectionel au moyen de deux programmes, `sender` et `receiver` . Ces deux programmes devront être produit au moyen de la cible par défault d'un Makefile. Cette implémentation DOIT fonctionner sur les ordinateurs **Linux** de la salle Intel, bâtiment Réaumur.

Les deux programmes nécessitent au minimum deux arguments pour se lancer :
`sender hostname port`, avec `hostname` étant le nom de domaine ou l'adresse IPv6 du receiver, et `port` le numéro de port UDP sur lequel le `receiver` a été lancé.
`receiver hostname port`, avec `hostname` le nom de domaine où l'adresse IPv6 sur laquelle le `receiver` doit accepter une connection (: : pour écouter sur toutes les interfaces), et `port` le port UDP sur lequel le `receiver` écoute.

Enfin, ces deux programmes supporteront l'argument optionnel suivant :
`-f/--filename X`, afin de spécifier un fichier X à envoyer par le `sender` ou le fichier dans lequel sauver les données reçues par le `receiver` . Si cet argument n'est pas spécifié, le `sender` envoye ce qu'il lit sur l'entrée standard (CRTL-D pour signaler EOF), et le `receiver` affiche ce qu'il reçoit sur sa sortie standard.

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 --filename fichier.dat ::1 12345`	Idem
`sender -f fichier.dat localhost 12345`	Idem
`receiver :: 12345 > fichier.dat 2> log.log`	Lance un receiver qui écoute sur toutes les interfaces, et affiche le contenu du transfert sur la sortie standard, qui est redirigée dans fichier.dat, alors que les messages d'erreur et de log sont redirigés dans log.log

3 Tests

3.1 INGInious

Des tests INGInious seront fournis chaque semaine pour tester différentes parties de vos implémentations. Si vous les réussissez durant la semaine, vous gagnez un point de réserve pour l'évaluation finale du 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 le documenter dans votre rapport.

3.3 Test d'interopérabilité

Vos programmes doivent être inter-opérables avec les implémentations d'autres groupes. Vous ne pouvez donc pas créer de nouveau type de segments, ou rajouter des méta-données dans le payload. Une semaine avant la remise du projet, vous devrez tester votre implémentation avec 2 autres groupes (votre `sender` et leur `receiver` , votre `receiver` et leur `sender`).

4 Planning et livrables

Première soumission, 23/10/2015 :

1. Rapport (max 4 pages, en PDF), décrivant l'architecture générale de votre programme, et répondant au minimum aux questions suivantes :
   - Comment avez-vous choisi la valeur du retransmission timeout ?
   - Quelle est la partie critique de votre implémentation, affectant la vitesse de transfert ?
   - Quelle stratégie de tests avez-vous utilisée ?
2. Implémentation des deux programmes permettant un transfert de données en utilisant le protocole
3. Suite de tests des programmes.
4. Makefile dont la **cible par défaut** produit les deux programmes dans le répertoire courant avec comme noms `sender` et `receiver` , et dont la **cible tests** lance votre suite de tests.

Tests d'inter-opérabilité, 29/10/2015, salle Intel, 14h-18h.

Deuxième soumission, 30/10/2015 :
Même critères que pour la première condition, le rapport devant décrire en plus le résultat des tests d'interopérabilité en annexe, ainsi que les changements effectués au code si applicable.

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

`/`	Racine de l'archive
`Makefile`	Le Makefile demandé
`src/`	Le répertoire contenant le code source des deux programmes
`tests/`	Le répertoire contenant la suite de tests
`rapport.pdf`	Le rapport

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 Moodle.

5 Evaluation

La note du projet sera composée des trois parties suivantes :

1. Implémentation (tests, qualité du code). Les points de cette partie peuvent être augmentés par les points de réserve des tests INGInious, sans pour autant dépasser le maximum ;
2. Rapport ;
3. Code review individuelle du codes de deux autres groupes. Effectuée durant la semaine suivant la remise du projet. Vous serez noté sur la pertinence de vos review.

6 Ressources utiles

Les manpages des fonctions suivantes est 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, shutdown, read, write, fcntl, getnameinfo, htonl, ntohl, ...`

[Sha15] est un livre en-cours de rédaction sur le C, couvrant de nombreux aspects du language à l'aide d'exemples, ainsi que d'autres sujets utiles, tels que techniques de debugging, unit-testing, analyse de performance.

[SR05] et [Ker10] sont deux livres de références sur la programmation système dans un environnement UNIX, disponibles en bibliothèque INGI.

[DC01] est le livre de référence sur les sockets TCP/IP en C, disponible en bibliothèque INGI.

[Shab] et [Shaa] sont deux mini-tutoriels pour la création d'un serveur et d'un client utilisant des sockets UDP.

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

Références

[Che+14] Peng Cheng et al. "Catch the Whole Lot in an Action : Rapid Precise Packet Loss Notification in Data Centers". In : *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. NSDI'14. Seattle, WA : USENIX Association, 2014, p. 17–28. ISBN : 978-1-931971-09-6. URL : https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-cheng.pdf.

[DC01] Michael J. Donahoo et Kenneth L. Calvert. *TCP / IP sockets in C, A practical guide for programmers*. 2001.

[Ker10] Michael Kerrisk. *The Linux programming interface : a Linux and UNIX system programming handbook*. 2010.

[Low] Spencer Low. *The World of select()*. URL : http://www.lowtek.com/sockets/select.html.

[Shaa] Graham Shaw. *Listen for and receive UDP datagrams in C*. URL : http://www.microhowto.info/howto/listen_for_and_receive_udp_datagrams_in_c.html.

[Shab] Graham Shaw. *Send a UDP datagram in C*. URL : http://www.microhowto.info/howto/send_a_udp_datagram_in_c.html.

[Sha15] Zed A. Shaw. *Learn C The Hard Way*. 2015. URL : `http://c.learncodethehardway.org/book/index.html`.

[SR05] W. Richard Stevens et Stephen A. Rago. *Advanced programming in the Unix environment*. 2005.