Projet – **Routage anonyme**

{Emmanuel.Godard, Yann.Esposito}@lif.univ-mrs.fr

janvier 2005

☞ *Le but de ce projet va être de réaliser des programmes suivant un protocole d'"Onion Routing" comme TOR http://tor.eff.org/.*

1 Principe

Lorsqu'un client veut se connecter à un serveur, celui-ci envoie sur le réseau des paquets contenant l'adresse du serveur ainsi que son adresse personnelle (ce qui lui permet de recevoir les paquets de retour). De plus toute personne écoutant la ligne peut savoir à qui se connecte le client et savoir ce qu'ils se disent. Nottament, la loi française oblige les fournisseurs d'accès à conserver les logs de connexions pendant deux ans. Ces logs doivent contenir au moins l'adresse source et de destination de chaque connexion. Allez vérifier sur le site de la CNIL (http://www.cnil.fr) les informations qui sont disponibles pour le serveur.

Pour de multiples raisons (voir le site de TOR) on peut souhaiter être anonyme, ou au moins qu'il soit plus difficile d'écouter la conversation entre un client et un serveur.

Pour cela, on peut utiliser un protocole qui va transférer tous les paquets sur un proxy intermédiaire qui se fera passer pour le client. Mais ce niveau de confidentialité est souvent trop faible. En effet, même si les connexions entre le client et le proxy sont chiffrées, il est toujours possible de retrouver votre chemin. Un tiers peu se faire passer pour le proxy de confiance, un tiers peut attaquer le proxy et écouter tout ce qui se passe. Il est facile en étudiant la quantité de débit de ce qui entre et qui sort du proxy de savoir quel client est connecté à quel serveur.

Il semble naturel de vouloir améliorer ce système en combinant plusieurs routeurs intermédiaires. Chacun de ces routeurs ne connaissant que son prédécesseur et son successeur. Seul le dernier des routeur pouvant avoir une version non chiffrée des données transmise (mais ignorant qui est le vrai client). Cette fois-ci l'analyse du réseau se révèlera beaucoup plus délicate.

La création d'un tel réseaux augmentant sensiblement l'anonymat se fera, dans ce projet, en plusieurs étapes :

1. **Création d'un contrôleur et d'un routeur n'utilisant pas de chiffrement des données** : Le programme de contrôle s'occupant de créer un chemin, le programme de routage se contentant de faire l'intermédiaire.
2. **Ajout du chiffrement symétrique** : ajout d'un service de la part des routeurs (envoi de la clé symétrique), le contrôleur chiffre plusieurs fois les données.
3. **Protocole de transfert de clé sécurisé** : utilisation de clés publiques pour le transfert des clés symétriques.
4. **Ajout d'un système publication des routeurs** : il faut que les contrôleurs puissent avoir accès aux adresses des routeurs.

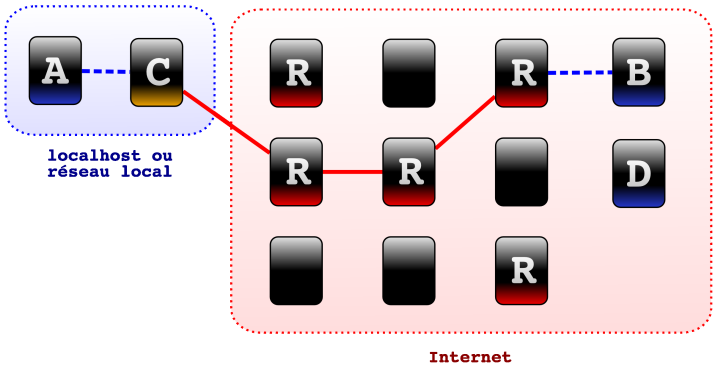
Pour entrer plus en détail voilà comment tout doit fonctionner :

Le contrôleur possède la liste des routeurs disponibles sur le réseau. Soit il l'obtient d'un serveur centralisé, soit il l'obtient de manière décentralisée en entrant en contact avec les routeurs déjà connus.

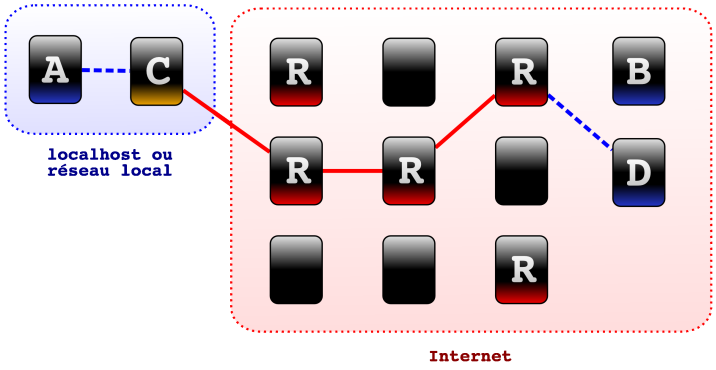
Dans un premier temps, on considèrera un moyen statique d'obtention de la liste des routeurs par exemple celle-ci sera contenue dans un fichier auquel le contrôleur aura accès.

Dans l'exemple que nous considérons, A est un client standart.

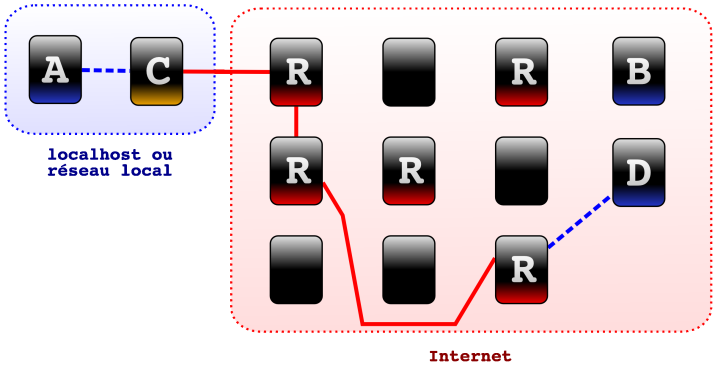
Lorsque A veut parler à B, le contrôleur établi un chemin chiffré entre les routeurs de manière à ce que le dernier routeur se comporte comme le client A.



Lorsque A veut se connecter à D, le chemin de routeur ne change pas nécessairement :



Au bout d'un certain temps, un nouveau chemin est établi par le contrôleur :



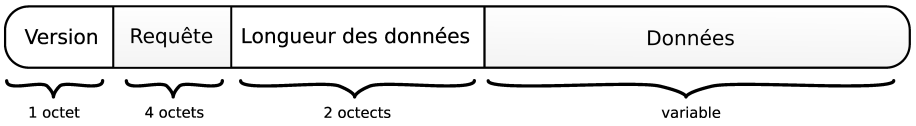
De plus le contrôlleur et les routeurs doivent pouvoir gérer plusieurs connexions simultanées.

2 Contrôleur, routeur et proxy HTTP

☞ *Le but de cette section va être de pouvoir parler à un serveur en faisant passer les paquet par l'intermédiaire de plusieurs routeurs. Pour cela, il est nécessaire d'avoir à sa disposition un contrôleur qui va choisir un chemin, puis encapsuler les données du client vers le premier routeur. Le rôle du routeur se contentant pour l'instant de jouer le rôle d'intermédiaire.*

☞ *Une attention toute particulière devra être apportée aux logs de vos serveurs. Il faut qu'ils soient de la meilleure qualité possible pour pouvoir débugger facilement. De plus un système devra exister qui permette de récupérer les logs de vos serveurs.*

☞ *Dans tout le projet, les paquets que vous allez traiter auront tous le format suivant :*



Nous considèrerons que la version sera `0`.

2.1 Routeur

☞ *Pour les utilisateurs de Python, l'utilisation de la librairie* `asyncore` *est autorisée et conseillée. Dans les autres langages, la methode par select devrait être la plus efficace mais souvent plus difficile à programmer. Quelquefois, selon le système d'exploitation, l'utilisation de* `fork()` *peut se révéler plus efficace que l'utilisation de threads. A vous de justifier votre choix de programmation.*

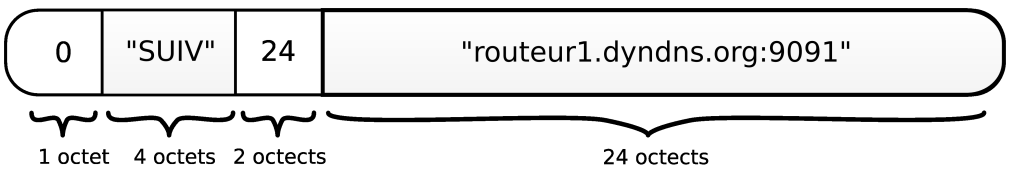
Celui-ci offre plusieurs services :
- demander d'ouvrir un flux vers le successeur ;
- demander d'ouvrir un flux vers un successeur identifié par un numero de session ;
- écrire sur le flux créé ;
- fermer le flux créé ;
- récupération des logs ;

2.1.1 Services proposés par le routeur

Les services que proposent le routeur sont accessibles par les clients avec les requêtes : `SUIV`, `CONN FLUX`, `FERM` et `BKDR`.

- `SUIV` : Le routeur se connecte alors sur le serveur situé à l'adresse et sur le port indiqué dans la partie donnée du paquet. Si la connexion réussie, le routeur renvoie au client `COOK` (COnnexion OK). Le routeur renvoi `ERRC` (ERReur Connexion) dans le cas où la connexion à échouée.
  À partir du moment où la connexion est acceptée, le routeur envoi vers son client tout ce qu'il reçoit de son successeur en encapsulant les données dans un paquet normalisé. La partie requête du paquet étant `DATA`.

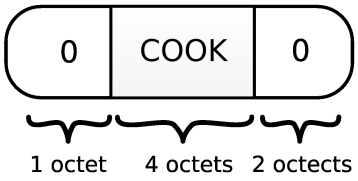
  Voici un exemple de paquet pouvant être reçu par le routeur :



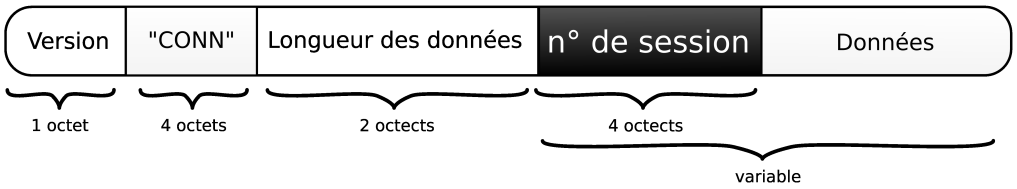
Par convention j'écris entre guillements (") les valeurs codées sous forme de chaîne de caractères et sans guillemet les valeurs décimales. Par exemple le paquet précédent se voit de la façon suivante :

```
Offset    ------------ notation hexadecimale -------------   ---- ASCII -----
00000000  00 53 55 49 56 00 18 72  6f 75 74 65 75 72 31 2e  | SUIV  routeur1.|
00000010  64 79 6e 64 6e 73 2e 6f  72 67 3a 39 30 39 31     |dyndns.org:9091|
```

Un paquet de retour possible est :



- `CONN` : ce service sera appelé lorsque le routeur sera en bout de chemin et devra se connecter à un serveur. En plus de l'adresse et du port, les données contiendront un numero de session. Lorsque le serveur enverra des données, il faudra encapsulées celles-ci dans un paquet contenant aussi le numéro de session. En pratique le paquet est le suivant :



  Les paquets de retours comme `COOK` et `ERRC` doivent aussi être modifiés de la même façon.
- `FLUX` : lorsque le routeur reçoit cette requête, il envoi le message contenu dans la partie données du paquet à son successeur. Si la connexion à été créée avec la requête `CONN` plutôt que `SUIV` alors les 4 premiers octects des données sont un numéro de session ce qui permet de savoir à quel serveur envoyer les données.
- `FERM` : Le routeur fermera la connexion vers son successeur.
- `BKDR` : pour BacKDoor, lorsqu'un client envoi cette requête, alors le routeur envoi l'ensemble de ses logs au client. Cette requête permettra de pouvoir faire du deboguage, mais peu aussi servir de backdoor pour savoir qui s'est réellemnt connecté avec qui.

Si le sucesseur d'un routeur ferme la connexion alors, le routeur envoie à son client utilisant ce successeur la commande `CLOS` (CLOSed). Si la connexion a été obtenue avec `CONN` plutôt que `SUIV` alors le paquet contiendra aussi le n° de session dans la partie données.

2.1.2 Contrôleur

☞ *Le contrôleur sera un serveur qui devra dans le cas général être lancé sur le même poste que le client en tant que démon. Il va se comporter soit comme le serveur cible soit comme un proxy* HTTP *. Le contrôleur joue un rôle similaire à celui des routeurs, mais en plus il contrôle la création des chemins et les encapsulation des données. C'est le contôleur qui va créer un* tunnel *de routeur. C'est au contrôleur de décider quelle sera la longueur des chemins et qui gèrera les numéros de session. Le contrôleur décidera aussi par la suite les clés à utiliser pour le chiffrement.*

Le contrôleur doit pouvoir fonctionner selon deux modes : un mode simulation de serveur et un mode proxy HTTP .

- **simulation de serveur**, l'adresse et le port du serveur cible[1] est donné directement en paramètre lors du lancement du contrôlleur.

[1] B sur les shémas de la section 1

– **proxy HTTP** , le serveur terminal est évalué lors des requêtes HTTP .

Le routeur doit posséder les fonction internes suivantes :
– `int nouveauChemin(int i)` : cette fonction va créer un nouveau chemin de longueur `i` . Pour cela, il a à sa disposition une liste de routeurs disponibles. Il va vérifier que la connexion se fait bien jusqu'au dernier serveur. Si la connexion s'est correctement établie, la fonction renvoie 0 sinon elle renvoie le numéro du routeur auquel la connexion à échouée.
– `supprimeChemin()` : va fermer la connexion entre tous les routeurs qui forment le chemin courant.
– `encapsule(string message)` : cette fonction encapsule les données de façon à ce que le message encapsulé puisse être envoyé au sucesseur direct et soit compris par le dernier routeur du chemin.

De plus, pour le début du projet, nous considérons que le contrôleur possède la liste des routeurs accessibles ; soit dans une structure de données remplie manuellement, soit dans un fichier qui sera chargé à chaque démarrage du contrôleur. Ajouter un système de publication des routeur fera partie des dernières étapes du projet.

La conception du contrôleur sera très proche de celle des routeurs. Voici les requêtes que peuvent formuler les clients au contrôleur et les services qui leur sont alors fournis :
– `CONN` : Demande au dernier routeur de se connecter à l'adresse indiquée dans la partie données. Si l'adresse ne contient pas de port, alors c'est le port 80 qui sera choisi par défaut. Une fois la connexion établie, le contrôleur reçoit les données encapsulées du serveur cible, les désencapsules et les envois vers son client.
– `FLUX` : le contrôleur encapsule les données de façon à ce que les données soient envoyées au serveur cible.
– `BKDR` : il faut traiter cette demande de la même façon que pour les routeurs en envoyant la totalité des logs au client qui fait cette requête.
– `<MOT CLE HTTP> http://adresse[:port]/chemin/vers/fichier ...\r\n` : où `<MOT CLE HTTP>` est l'un des mots clé :
  – `GET`
  – `HEAD`
  – `POST`
  – `PUT`
  – `DELETE`

  Fait la même chose que `CONN` suivit de la requête `FLUX` qui transmet l'ensemble de la requête au serveur cible. Cette dernière requête est utile pour pouvoir utiliser un navigateur internet à travers le réseaux. Il suffira d'utiliser le contrôleur comme un proxy HTTP .

2.2 Résumé et exemple

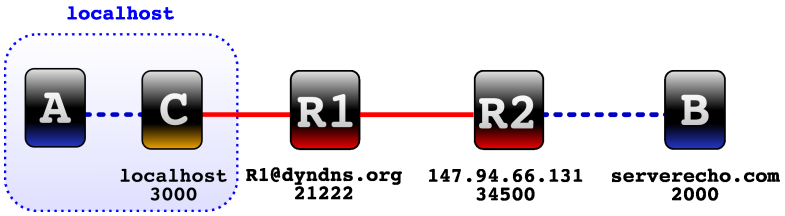
Pour résumer voici la table qui résume l'ensemble des requêtes pour les routeurs et les contrôleurs :
Pour le routeur :

Requête	Données	Retour requête
`SUIV`	`"adresse:port"`	`"COOK"` ou `"ERRC"`
`CONN`	`n°session"adresse:port"`	`"COOK"` ou `"ERRC"`
`FLUX`	données brutes	-
`FERM`	-	`"CLOS"`
`BKDR`	-	logs

Pour le contrôleur :

Requête	Données	Retour requête
`CONN`	`n°session"adresse:port"`	`"COOK"` ou `"ERRC"`
`FLUX`	données brutes	-
`BKDR`	-	logs
Requête proxy	-	-

Pour fixer les idées nous donnons un exemple des paquets passant sur le réseaux. Le réseau concerné est le suivant :



- `A` est un client echo ;
- `B` est un serveur echo d'adresse `serverecho.com:2000` ;
- `C` est le contrôleur d'adresse `localhost:3000`. Il a été lancé en mode simulation ; c'est-à-dire avec la ligne de commande : `controleur 3000 serverecho.com 2000` ;
- `R1` est le premier routeur d'adresse `R1@dyndns.org:21222` ;
- `R2` est le second routeur d'adresse `147.93.66.131:34500` ;

Sont écrit entre parenthèse les commande qui ne nécessitent pas d'entrée ou de sortie sur les sockets comme les connexions et les "close". Les paquets sont écrit selon la notation suivante : `v."req".l."data"` avec :

- `v` le numero de version en décimal (sur 1 octect) ;
- `"req"` la requête codée en ASCII (sur 4 octects) ;
- `l` la longueur en décimal (sur 2 octects) ;
- `"data"` les données codée en ASCII (sur l octects) ou `nombre"data"` ou nombre correspondra au numéro de session en décimal (4 octects) ;

Voir la figure 1 page suivante.

```
--- Creation du tunnel avant la connexion du client ---
C  -> R1 : (connect)
R1 -> C  : (accept)
C  -> R1 : 0."SUIV".19."147.94.66.131:34500"
R1 -> R2 : (connect)
R2 -> R1 : (accept)
R1 -> C : 0."COOK".0.
--- Creation de la connexion ---
A  -> C  : (connect)
C  -> A  : (accept)
C  -> R1 : 0."FLUX".30.[0."CONN".23.2345"serverecho:2000"]
R1 -> R2 : 0."CONN".23.2345"serverecho:2000"
R2 -> B  : (connect)
B  -> R2 : (accept)
R2 -> R1 : 0."COOK".4.2345
R1 -> C  : 0."DATA".11.[0."COOK".4.2345]
--- Envoi des données vers le serveur ---
```

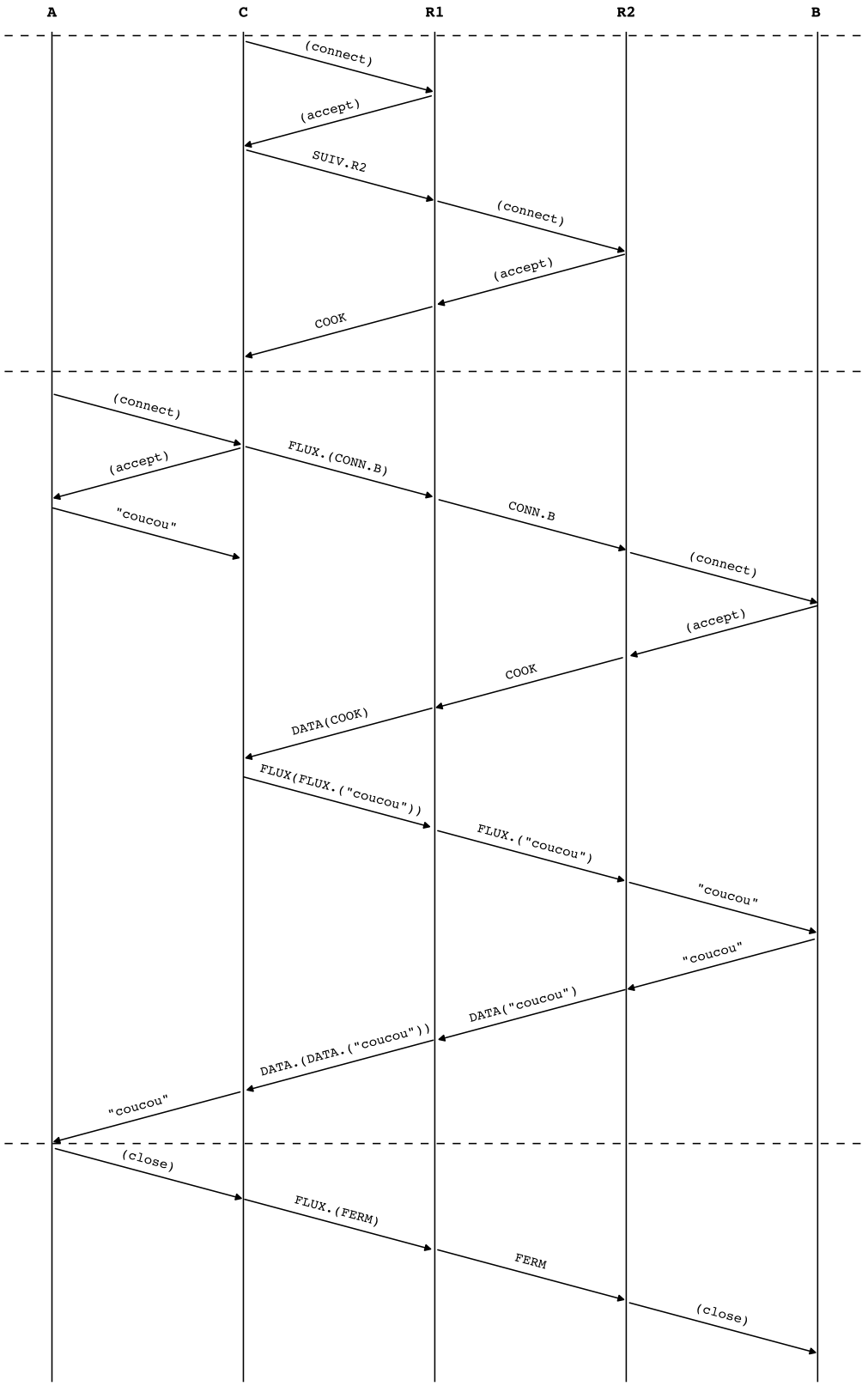


Fig. 1 – Illustration de la connexion echo

```
A  -> C  : 0."FLUX".6."coucou"
C  -> R1 : 0."FLUX".17.[0."FLUX".10.2345"coucou"]
R1 -> R2 : 0."FLUX".10.2345"coucou"
R2 -> B  : "coucou"
B  -> R2 : "coucou"
R2 -> R1 : 0."DATA".10.2345"coucou"
R1 -> C  : 0."DATA".17.[0.DATA.10.2345"coucou"]
C  -> A  : "coucou"
--- Fermeture de la session ---
A  -> C  : (close)
C  -> R1 : 0."FLUX".11.[0."FERM".4.2345]
R1 -> R2 : 0."FERM".4.2345
R2 -> B  : (close)
```

2.1. Écrivez le même type de conversation en utilisant le même réseaux mais en ajoutant un deuxième client echo `D` qui se connecte en même temps à un autre serveur echo `E`.

(a) Pourraient-ont se passer des numéros de session ? Si oui, quelles fonctionnalités du réseau anonyme perdraient-on ? Si non, expliquez en quoi ces numéros de sessions sont absolument nécessaires à la bonne utilisation du réseau anonyme.

Voici un autre exemple de dialogue possible entre un client HTTP `A` et un serveur HTTP `B` qui se parlent par l'intermédiaire d'un contrôlleur `C` et de deux routeurs `R1` et `R2`. L'adresse IP de `R2` est `147.94.66.131` l'adresse de `B` est `servhttp.com` et écoute sur le port 80.

```
C  -> R1 : 0."SUIV".19."147.94.66.131:34500"
R1 -> R2 : (connect)
R2 -> R1 : (accept)
R1 -> C  : 0."COOK".0
A  -> C  : GET http://servhttp.com HTTP/1.0\r\n
C  -> R1 : 0."FLUX".26.[0."CONN".19.456"servhttp.com:80"]
R1 -> R2 : 0."CONN".19.456"servhttp.com:80"
R2 -> B  : (connect)
B  -> R2 : (accept)
R2 -> R1 : 0."COOK".4.456
R1 -> C  : 0."DATA".11.[0."COOK".4.456]
C  -> R1 : 0."FLUX".49.[0."FLUX".42.456"GET http://servhttp.com HTTP/1.0\r\n"]
R1 -> R2 : 0."FLUX".42.456"GET http://servhttp.com HTTP/1.0\r\n"
R2 -> B  : "GET http://servhttp.com HTTP/1.0\r\n"
B  -> R2 : HTTP/1.0 200 OK ...
R2 -> R1 : 0."DATA".231.456"HTTP/1.0 200 OK ... "
R1 -> C  : 0."DATA".238.[0."DATA".231.456"HTTP/1.0 200 OK ... "]
C  -> A  : "HTTP/1.0 200 OK ... "
B  -> R2 : (close)
R2 -> R1 : 0.CLOS.4.456
R1 -> C  : 0."DATA".11.[0."CLOS".4.456]
C  -> A  : (close)
```

Remarquez que la connexion entre `R1` et `R2` n'a pas été fermée.

2.2. Dans un premier temps programmez un routeur sans gérer la requête `CONN`.

(a) En utilisant telnet, vérifiez que vous pouvez vous connecter à travers plusieurs routeurs. Pour cela utilisez comme destination un serveur echo.

2.3. Programmez un contrôleur qui ne gére pas `CONN` et vérifiez que tout fonctionne. En particulier vérifiez, en utilisant ethereal, que les message sont bien ceux attendus. Essayez l'ensemble des cas suivants :

(a) contrôleur en mode simulation vers un serveur echo en utilisant un, deux et trois routeurs.

2.4. Ajoutez le service `CONN` au routeur et au contrôleur. Vérifiez que tout fonctionne correctement :

(a) Soit le tunnel suivant : un contrôleur sur `127.0.0.1:9090`, le routeur sur `127.0.0.1:9091`, un serveur echo sur `127.0.0.1:3000`. En utilisant ethernet donnez les trames qui correspondent à la connexion de deux clients echo qui se connectent au serveur echo via le tunnel.

(b) Recommencez en ajoutant un second routeur, puis un troisième (toujours sur l'ordinateur local).

(c) Vérifiez que les connexions fonctionnent toujours même lorsque les ordinateurs sont différents.

2.5. Ajoutez la fonctionnalité mode proxy HTTP au contrôleur.

(a) Vérifiez en vous connectant via le contrôleur vers votre site personnel ou sur n'importe quel site hébergé par *daumier*. N'oubliez pas de vérifier avec ethereal que les paquets sont bien envoyés via le contrôleur.

3 Chiffrement symétrique

☞ *Après avoir réalisé l'étape précédente, il est impossible pour le serveur de connaître la véritable adresse IP du client. De plus, les log conservés par les fournisseurs d'accès devraient déjà ne plus posséder l'adresse IP du vrai serveur.*
Cependant, toute personne écoutant la ligne au niveau du fournisseur d'accès (entre le contrôleur et le premier routeur) peut tout connaître de la connexion. Pour éviter cela, nous allons chiffrer les données.

Dans un premier temps, il faut ajouter une requête sur les routeurs :

`CSYM` : lorsque le routeur reçoit cette requête, il récupère la clé dans la partie données du paquet et l'utilisera pour déchiffrer les messages qui lui parviendront par la suite.

Plus précisément le paquet est : `0."CSYM".32."clé symmétrique"`

Normalement, vous devriez posséder une fonction `encapsule` et une fonction `desencapsule` (ou `decapsule` pour les buveurs de bière) disponible par le contrôleur et sur les routeurs. Ces fonctions ajoutant ou supprimant les entêtes. Il suffit de modifier ces fonctions pour prendre en compte le chiffrement des données. Attention les données que renvoient les routeurs devront maintenant être chiffrées avec cette clé symmétrique.

Vous utiliserez l'algorithme AES 128 comme méthode de chiffrement. Même si c'est la méthode de chiffrement la plus lente, utilisez la méthode de chiffrement par flux plutôt que par blocs.

Si vous ne possédez pas de bibliothèque AES, utilisez le chiffrement XOR dans ce cas demandez l'autorisation à votre responsable de TP.

L'utilisation de librairies est vivement conseillée et préférée même si la programmation de cet algorithme peut-être un bon exercice pour sa culture personnelle. Si un message doit traverser n routeurs alors, celui-ci sera chiffré n fois par le contrôleur avant d'être envoyé.

Prenez gare à l'odre dans lequel vous chiffrez les données.

3.1. Vérifiez les messages circulant sur le réseau après avoir programmer cette étape.

4 Transfert de clé sécurisé

☞ *Après la dernière étape, toutes les connexions sont chiffrées à l'exception du passage de clé symmétrique. Cette faille permettant à toute personne d'écouter la ligne entre le contrôlleur et le premier routeur de reconstruire toutes les communications. Il faut donc avoir un moyen d'envoyer cette clé symmétrique sans risque. Pour cela, nous allons utiliser un chiffrement asymétrique.*

Nous considérons que, en plus des adresses des routeurs, le contrôleur possède aussi les clés publiques de ceux-ci.

Les routeurs récupèrent une clé qui a été chiffrée avec leur clé publique. Le nouveau paquet est donc :

`0.CSYM.32."clé chiffrée avec la clé publique du routeur"`

L'algorithme asymétrique à utiliser sera RSA. Comme précédemment l'utilisation de librairies est à préférer à la programmation personnelle de cet algorithme de chiffrement.

5 Réflexions sur les attaques possibles du réseau

☞ *Vous avez réalisé un réseau censé rendre anonyme ces utilisateurs. Mais l'anonymat n'est pas une notion binaire. Quel est le réel degrés d'anonymat que procure ce réseau ?*
Par attaque, on entend toute manière de récupérer des informations censées êtres cachées par le réseau.
Dans toutes les questions de cette section veillez à bien précisez quels moyens les attaquants doivent posséder ; qu'ils soient matériels (cluster, superordinateurs) ou théoriques (connaître une partie du message).
Veillez aussi à proposer des améliorations du réseau qui permettent à celui-ci de s'améliorer et de contrer ces attaques. De plus donnez les implications en terme d'efficacité du réseau de telles améliorations.
Soyez aussi précis que possibles et si possibles donnez un exemple concret d'attaque. Vous pourrez par exemple donner la suite des paquets récupérés sur le net qui peuvent servir.

5.1. Est-il possible d'attaquer en écoutant sur la ligne entre le contrôleur et le premier routeur ?

5.2. Est-il possible d'attaquer en écoutant sur la ligne entre le dernier routeur et le serveur ?

5.3. Quelles informations un attaquant possédant la possibilité d'écouter sur toutes les lignes publiques peut-il récupérer ?

5.4. Quelles informations un routeur malveillant peut récupérer ?

5.5. Quelles informations un attaquant possédant plusieurs routeurs malveillants peut posséder ?

5.6. Y-a-t'il un moyen d'être certain de ne pas parler à un routeur malveillant ?

5.7. Quelles attaques supplémentaires existent ?

5.8. Faites un résumé succints des hypothèses qui permettent d'assurer l'anonymat des utilisateurs.

Si vous êtes arrivé jusqu'ici ; félicitations ! Vous avez terminé le projet demandé. Si vous voulez aller plus loin, lisez la suite (les points sont des bonus). Bien entendu vous ne pouvez pas avoir plus que 20 et n'oubliez pas que le projet ne compte que pour une partie de votre note finale. Ne vous épuisez pas inutilement, la suite du projet est sans fin. Faites ce qui vous intéresse. Bonne chance !

6 Améliorations

☞ *Si vous êtes arrivé au bout du projet et que vous voulez l'améliorer voici quelques idées. En aucun cas ne faites une amélioration si vous n'avez pas fini le projet.*

6.1 Publication des routeurs

6.1. Programmez un serveur qui centralise la liste des routeurs ainsi que leur clé publique.

(a) Programmez un serveur qui peut inscrire et désinscrire des routeurs. Faites en sorte que le serveur possède aussi une requête qui permet au contrôleurs de récupérer l'ensemble des routeurs inscrits.

(b) Améliorez le programme en associant à chaque routeur inscrit sa clé publique.

(c) Améliorez le programme en utilisant un système de certificat qui permet d'assuré aux contrôleur que le serveur est le bon.

6.2. Ajoutez un système de requêtes qui permet aux routeurs de se publier à d'autres routeurs de façon décentralisée.

6.3. Quel système de publication vous semble le plus sécurisant pour les utilisateurs du réseaux ? Pourquoi ?

6.2 Améliorations et questions en vrac

1. contrôle de la quantité de flux ;
2. routage plus intelligent ;
3. utilisation de plusieurs chemins ;
4. encapuslation des connexions avec SSL ;
5. ajout de code C, C++ ou OCamL pour les fonctions les plus internes. ;
6. ajout d'un temps d'attente aléatoire d'envoi des paquets. ;
7. choix des positions des routeurs (debut, fin ou milieux) ;
8. programmer le routeur pour être un proxy SOCKS 5 ou un proxy SOCKS 4 avec sécurisation du DNS (privoxy). ;
9. expliquez comment mettre en place un système de rendez-vous qui permet de cacher des services et réalisez-le ;