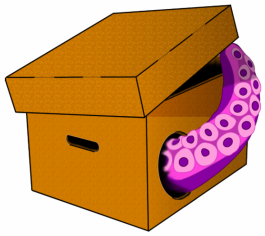

tNETacle

Documentation Technique

Ce document a pour but de mettre a disposition des informations techniques à propos du tNETacle et de ces composants. Il contient la documentation des API publiques ainsi que la description des mechanismes principaux du tNETacle. Tous les dévelopeurs doivent l'avoir lu avant de contribuer du code au projet.

(| €~

Titre	Documentation technique
Date	17/01/2013
Auteurs	tNETacle
email	team@tnetacle.org
Sujet	Ce document est la documentation technique du projet tNETacle. Il est destiné à quiconque voudrait rejoindre le projet
Version	9

Version	Date	Auteur	Section	Commentaires
0	23/03/2012	Tristan Le Guern	Document entier	Importation des documents du TA2
1	23/03/2012	Tristan Le Guern	1.2	Mise à jour de la libtuntap
2	23/03/2012	Tristan Le Guern	1.1	Mise à jour de la libtclt
3	23/03/2012	Nicolas Vivet	2.4 and 5	Améliore la section sur la libevent et les bogues
4	23/03/2012	Antoine Marandon	2.1	Mise à jour de l'API de log
5	23/03/2012	Tristan Le Guern	2.1	Uniformisation de l'API de log
6	23/03/2012	Antoine Marandon	3	Ajout du Pipeline et de la Paquet Factory
7	17/05/2012	Antoine Marandon	4	Ajout des méta-connexions
8	18/05/2012	Nicolas Vivet	Abstract	Ajout d'un abstract
9	17/01/2013	Florent Vivet	Libtclt	Lien vers la documentation sur le phabricator

Contents

List of Figures

1 About this document

Ce document est fait pour indiquer la meilleur façon d'utiliser nos APIs. Pour cela, nous assumons néanmoins que:

- Vous êtes familier avec le langage C;
- Vous êtes familier avec la programation réseau;
- Vous êtes habitué à travailler avec plusieurs système d'exploitation;

2 Introduction

Le tNETacle est un logiciel de réseau privé virtual décentralisé sécurisé, simple d'usage and portable.

En plus de son chiffrement fort, le tNETacle utilise la séparation de privilège. Cela veut dire que tous les éléments priviligiés sont divisés utilisant les fonctions socketpair() et fork(). Le processus principal utilise la fonction chroot() pour tourner dans un environnement restreint. Il sera responsable de la boucle principale et communiquera avec de plus petit processus, qui gardent leurs privilèges, pour des opérations plus spécifiques.

Le tNETacle fonctionne directement. Il ne nécessite pas l'édition de fichier compliqués, ni de manipuler la ligne de commande: tout est configuré par défaut sans manipulation. Contrairement à d'autres logiciels VPN, notre interface utilisateur est séparée du processus principal. C'est un concept important puisqu'il permet la création de différents clients avec une interface puissante comme: un client console (CLI), une interface web (WebUI), ou une interface graphique (GUI).

Le tNETacle doit tourner partout dans le but de connecter tout le monde, quelquesoit l'équipement utilisé. C'est pourquoi, nous sommes compatible avec Windows NT, Mac OS X, Linux, FreeBSD, OpenBSD et NetBSD. Ces systèmes d'exploitations sont nos cibles officielles, cependant il n'y a pas de raison pour que le tNETacle ne fonctionne pas sur d'autres UNIX-like: il suffit d'un compilateur C et d'un pilote tun.

Le code source est organisé en plusieurs parties: les bibliothèques indépendantes, le "core", et le client officiel.

3 Bibliothèques logicielles

Voici une liste de bibliothèque logicielles développées conjointement au tNETacle. Ces bibliothèques sont exportés séparément afin que d'autres projets puissent les utiliser.

3.1 Libtclt

La bibliothèque libtclt est faite pour faciliter le développement de client, graphique ou en ligne de commande, pour le tNETacle. Elle est décrite à l'adresse suivante : http://trac.medu.se/w/projects/libtclt_api/.

3.1.1 Préliminaires

La libtclt a pour but d'être simple d'utilisation et portable.

3.1.2 Instancier la bibliothèque

Avant toute chose vous devez instancier la bibliothèque en appelant `tnt_tclt_init()`. Cela initialisera les variables globales.

Interface

```
void tnt_tclt_init(void);
```

3.1.3 Libérer la bibliothèque

Lorsque vous en avez fini avec la bibliothèque, lorsque votre application quitte par exemple, vous pouvez appeler `tnt_tclt_destroy()` afin de libérer la mémoire.

Interface

```
void tnt_tclt_destroy(void);
```

3.1.4 Récupérer la version de la bibliothèque

Afin de pouvoir faire des mises à jour sans rendre tous les programmes incompatibles, la bibliothèque libtclt permet de requérir son numéro de version avec `tnt_tclt_get_version()`.

Interface

```
void tnt_tclt_get_version(void);
```

Cette bibliothèque est encore en stade de développement.

3.2 Libtuntap

La libtuntap est une bibliothèque développée pour faciliter la configuration de périphérique réseau virtuel de type `TUN` et `TAP` de façon portable. Elle permet aux applications utilisateurs de recevoir et d'envoyer des paquets réseaux.

Nous avons besoin de ces fonctionnalités pour développer le réseau virtuel.

3.2.1 Préliminaires

lintuntap est développée avec deux objectifs en tête:

- Portabilité - Un programme écrit avec la libtuntap devrait fonctionner sur tous les Systèmes d'Exploitation supportés.
- Simplicité - La bibliothèque fournie des API de configuration des périphériques virtuels, rien de plus.

3.2.2 Instancier un périphérique virtuel

La fonction `tnt_tt_init()` alloue et retourne un périphérique virtuel non-configuré. Si une erreur intervient lors de la création, la fonction retourne `NULL`.

Interface

```
struct device *tnt_tt_init(void);
```

3.2.3 Libérer un périphérique virtuel

Lorsque vous en avez fini avec le périphérique, vous pouvez appeler `tnt_tt_destroy()` afin de le détruire et de libérer la mémoire qui lui est associée.

Interface

```
void tnt_tt_destroy(struct device *);
```

3.2.4 Configurer un périphérique virtuel

Après avoir créé un périphérique avec `tnt_tt_init()`, vous devez le configurer en appelant `tnt_tt_start()`. Cette fonction permet de donner les paramètres par défaut à votre périphérique.

Le deuxième paramètre correspond au choix du comportement de votre device:

- mode tunnel (couche 3 du modèle OSI)
- mode ethernet (couche 2 du modèle OSI)

Le troisième paramètre correspond au numéro de votre périphérique, si cette fonctionnalité est supporté par votre Système d'Exploitation. Si vous n'avez pas besoin de le préciser, utilisez `TNT_TUNID_ANY`.

Si vous voulez complètement modifier la configuration de votre périphérique, et plutôt que d'en instancier un nouveau, vous pouvez en recycler un.

Pour cela, il suffit d'appeler `tnt_tt_stop()` qui s'occupera de déconnecter le périphérique et de le réinitialiser. Cette fonction est appelée automatiquement par `tnt_tt_destroy()`.

En cas d'erreur, `tnt_tt_start()` retourne `-1`, mais `tnt_tt_stop()` n'échoue jamais.

Interface

```
#define TNT_TUNMODE_ETHERNET 1
#define TNT_TUNMODE_TUNNEL 2

int tnt_tt_start(struct device *, int, int);
void tnt_tt_stop(struct device *);
```

3.2.5 Assigner le MTU

La libtuntap supporte la modification du 'Maximum Transmission Unit', aussi appelé MTU. Il n'y a pas de restriction sur la valeur passée a `tnt_tt_set_mtu()`, il est de la responsabilité du développeur de ne pas utiliser une valeur trop faible.

En cas d'erreur, `tnt_tt_set_mtu()` retourne `-1`, mais `tnt_tt_get_mtu()` n'échoue jamais.

Interface

```
int tnt_tt_get_mtu(struct device *);
int tnt_tt_set_mtu(struct device *, int);
```

3.2.6 Assigner une adresse MAC

La libtuntap supporte la modification de l'adresse MAC d'un périphérique virtuel.

Le second paramètre de `tnt_tt_set_hwaddr()` peut être n'importe quel adresse MAC valide ou la chaise de caractères `"random"`.

En cas d'erreur, `tnt_tt_set_hwaddr()` retourne `-1`, mais `tnt_tt_get_hwaddr()` n'échoue jamais.

Interface

```
char *tnt_tt_get_hwaddr(struct device *);
int tnt_tt_set_hwaddr(struct device *, const char *);
```

3.2.7 Assigner une adresse IP

`tnt_tt_set_ip()` gère aussi bien l'IPv4 que l'IPv6. Le premier paramètre est une adresse IP valide, le second un masque de sous réseau, tout deux en notation CIDR.

Seul le dernier appel à cette fonction est pris en compte. En cas d'erreur, `tnt_tt_set_ip()` retourne `-1`.

Interface

```
int tnt_tt_set_ip(struct device *, const char *, const char *);
```

3.2.8 Connecter le périphérique virtuel

Lorsque votre périphérique est correctement configuré, vous pouvez le connecter à l'aide de `tnt_tt_up()`.

Le périphérique sera près à l'emploi.

Interface

```
int tnt_tt_up(struct device *);
```

3.2.9 Déconnecter le périphérique virtuel

Vous pouvez déconnecter le périphérique virtuel avec `tnt_tt_down()`.

La configuration ne sera pas affecté, mais le périphérique ne recevra plus aucun paquet réseau avant un nouvel appel à `tnt_tt_up()`.

Interface

```
int tnt_tt_down(struct device *);
```

3.2.10 Exemple

Voici un exemple d'utilisation de la libtuntap sous licence ISC.

```
#include <sys/types.h>
#include <sys/socket.h>

#include <netinet/in.h>
#include <arpa/inet.h>

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <getopt.h>
#include "tun.h"

void
usage(void) {
        fprintf(stderr, "usage: -a addr/netmask -i interface -h");
        exit(1);
}

int
main(int argc, char *argv[]) {
        struct device *dev;
        int ch;
        char *addr;
        char *netmask;
        char *ptr;
        int interface;
        struct timeval tv;

        interface = TNT_TUNID_ANY;
        addr = "1.2.3.4";
        netmask = "255.255.255.0";
        while ((ch = getopt(argc, argv, "a:i:h:")) != -1) {
                switch (ch) {
                case 'a':
```

```
                        ptr = strchr(optarg, '/');
                        if (ptr == NULL)
                            usage();
                        *ptr = '\0';
                        ptr++;
                        addr = strdup(optarg);
                        netmask = strdup(ptr);
                        break;
                case 'i':
                        interface = atoi(optarg);
                        break;
                case 'h':
                default:
                        usage();
                }
        }

        dev = tnt_tt_init();
        if (tnt_tt_start(dev, TNT_TUNMODE_ETHERNET, interface) == -1)
            fprintf(stderr, "Can't start the device\n");
        printf("Interface: %s\n", tnt_tt_get_ifname(dev));

        if (tnt_tt_set_ip(dev, addr, netmask) == -1)
            fprintf(stderr, "Can't set ip\n");

        if (tnt_tt_up(dev) == -1)
            fprintf(stderr, "Can't bring interface up\n");

        tv.tv_sec = 5;
        tv.tv_usec = 0;
        select(1, NULL, NULL, NULL, &tv);

        if (tnt_tt_down(dev) == -1)
            fprintf(stderr, "Can't bring interface down\n");
        tnt_tt_stop(dev);
        tnt_tt_destroy(dev);
        return 0;
}
```

3.3 Containers

Nous développons pour le tNETacle une bibliothèque de containers. L'un de nos membre a l'a nommé "calm-containers".

Les principaux but des "calm containers" sont d'être léger, proche en performance de ce qu'on peut trouver dans les autres implementation et bien sûr d'être facile d'utilisation.

L'interface de programmation est relativement proche de celle des std::vector de la STL de C++.

Les principales différences entre "calm-containers" et ce qu'on trouve generalement sur le net sont:

- L'interface de programmation n'est pas en lettres majuscules car
- ce ne sont pas des macro fonction, en conséquence il est
- facile de debuger, car toutes les fonction, procédure et types existent pour de vrai. Il n'y a pas un seul `void *` dans le code.

3.3.1 vector

Les vecteurs sont les plus simple type de container. Il sont juste un simple tableau à une dimension d'éléments. Les accès aléatoire ainsi que le parcourt séquentiel sont rapides, et les vecteurs sont peu consommateur de mémoire. La taille des métadonnés pour les nos vecteur est de 12 octet sur un système 32bits et de 24 octet sur un système 64bits (deux size_t et un pointeur) ce qui est petit.

Comme nous parlons de containers génériques sans `void *`, nous avons besoins de générer autant de fonction qu'il y a de différents type a contenir. Ceci nous amène à un problème connu, comment nommer nos fonction générées pour qu'elles ne collisionnent pas entres elles. Nous avons choisi de laisser le choix au développeur, lui permettant de préciser le préfixe souhaité grâce à la macro VECTOR_PREFIX et le type à contenir grâce à VECTOR_TYPE.

```
#define VECTOR_TYPE foo
#define VECTOR_PREFIX bar
#include "vector.h"
#undef VECTOR_TYPE
#undef VECTOR_PREFIX
```

N'oubliez pas de supprimer les macro valeurs VECTOR_TYPE et VECTOR_PREFIX, sinon elles collisionneront si vous utilisez `vector.h` plusieurs fois d'affilée.

Une fois `vector.h` inclut, les fonctions suivantes seront générés.

```
struct vector_bar {
  foo *vec; /*the pointer to the allocated memory*/
  size_t size; /*the size of the vector in element*/
  size_t alloc_size; /*you'd better not touch this value*/
};

static inline foo *v_bar_end(struct vector_bar *v);
static inline foo *v_bar_next(foo *it);
```

```
static inline void v_bar_erase(struct vector_bar *v, foo *ptr);
static inline void v_bar_erase_range(struct vector_bar *v, foo *,
                                     foo*);
static inline foo v_bar_at(struct vector_bar *v, size_t i);
static inline foo *v_bar_atref(struct vector_bar *v, size_t i);
static inline foo v_bar_front(struct vector_bar *v);
static inline void v_bar_init(struct vector_bar *v);
static inline void v_bar_push(struct vector_bar *v, foo *val);
static inline int v_bar_resize(struct vector_bar *v, size_t);
static inline void v_bar_insert_range(struct vector_bar *, foo *,
                                      foo *, foo *);
static inline void v_bar_insert(struct vector_bar *, foo *, foo *);
static inline void v_bar_pop(struct vector_bar *v);
static inline void v_bar_delete(struct vector_bar *v);
static inline foo *v_bar_begin(struct vector_bar *v);
static inline foo v_bar_back(struct vector_bar *v);
static inline foo *v_bar_frontref(struct vector_bar *v);
static inline foo *v_bar_backref(struct vector_bar *v);
static inline foo *v_bar_find_if(struct vector_bar *v, foo *val,
                                 int (*)(foo const *, foo const *));
```

4 Modules

Les modules sont de petites portions de code qui ne sont pas exportées à l'extérieur de tNETacle. Leur but est d'encapsuler les dépendances externes telles que `zlib`, ou de proposer des fonctions utiles.

4.1 Compression - zlib

4.1.1 Dépendance

Pour la compression, nous utilisons la bibliothèque `zlib`. Cette bibliothèque a pour particularité d'être reconnue pour sa stabilité et sa fiabilité. De plus, elle n'est pas soumise à divers brevets.

Nous l'avons encapsulé ces fonctionalités dans le module 'src/compress.c'.

4.1.2 Compression

Vous pouvez compresser toute donnée en utilisant `tnt_compress()`. Cette fonction prend un `uchar *` (une chaîne de `unsigned char`), la taille associée à cette chaine, et la taille de la chaine compressée (valeur définie par la fonction appelée).

Elle retourne la chaine compressée ou `NULL` en cas d'erreur.

Interface

```
typedef unsigned char uchar;

uchar *tnt_compress(uchar *, const size_t, size_t *);
```

4.1.3 Décompression

Vous pouvez décompresser n'importe quelle donnée précédemment compréssée avec `tnt_compress()` grâce à l'aide de `tnt_uncompress()`.

Elle prend la chaine de caractère à décompresser, sa taille et la taille originelle de la chaine non-compréssée.

Interface

```
uchar *tnt_uncompress(uchar *, const size_t, const size_t);
```

4.2 Log

Afin de collecter des messages d'erreur dans un environment sans interface, nous avons développé une API de logging.

En mode debug, les messages seront écrit sur la sortie standard ou dans une console, en temps normal il seront écrit dans l'utilitaire 'syslog' s'ils est géré par votre Système d'Exploitation.

4.2.1 Setting up the API

Avant tout autre appel, vous devez initialiser la bibliotheque grâce à `log_init()`. Cette fonction va initialiser les variables globales, ouvrir les fichiers nécessaires et activer les routines initiales.

Interface

```
void log_init(void);
```

4.2.2 Assigner un préfix

Afin de différencier les messages écris par différent processus ayant le même nom, il est possible d'assigner un préfix qui sera affiché avant chaque message.

Interface

```
void log_set_prefix(char *);
```

4.2.3 Ecrire un message

L'API de loggin permet d'écrire des messages d'Erreur avec l'indice d'erreur du système, des messages d'Erreur simple, des Warning avec l'indice d'erreur du système, des Warning simple, des Notifications (indiquant quelque chose qui n'est pas un problème mais devrait peut être être adréssée), des Information et des messages de Debug.

Les deux functions d'Erreur prennent en premier paramètre une valeur numérique qui sera passée à `exit()`.

```
void log_err(int, const char *, ...);
void log_errx(int, const char *, ...);
void log_warn(const char *, ...);
void log_warnx(const char *, ...);
void log_notice(const char *, ...);
void log_info(const char *, ...);
```

```
void log_debug(const char *, ...);
```

4.3 Chiffrement - OpenSSL

4.3.1 Dépendance

Pour le chiffrement, nous utilisons la bibliothèque `OpenSSL`.

OpenSSL est une implémentation libre des protocoles SSL et TLS, reconnue pour sa stabilité et sa fiabilité.

Cette abstraction est placée dans un module plutôt qu'une bibliothèque pour la simple raison que les chances de changer cette dépendance sont faibles: il n'y a pas d'autres bibliothèques utilisable par le TtNETacle.

Il n'y a aucune API publique pour le moment.

4.4 Gestion des événements

4.4.1 Dépendance

Pour la gestion des événements systèmes (réseaux, IPC, ...), nous utilisons bibliothèque `Libevent`. Connue pour être stable et performante, la libevent 2 est utilisée par de nombreux projets libres et commerciaux.

Il n'y a aucune API publique pour le moment, mais elle gérera les versions 1 et 2 de la libevent.

5 Core

5.1 Utilité

Le Core correspond au daemon réseau utilisant les bibliothèques, API et modules cités plus haut. C'est lui qui est responsable de la communication au sein du VPN.

Il sera configurable grace à une interface spéciale appelée Client, qui communiquera avec lui grace à un socket réseau ou local.

Afin d'assurer une décentralisation, chaque utilisateur installer ce Core qui contiendra toute la logique du systeme.

Chaque core sera maître de son propre réseau et possèdera des politiques d'accessibilité par rapport aux autres noeuds. Il s'agit d'une application pair-à-pair.

5.2 Architecture

5.2.1 Séparation de privilège

Afin d'améliorer sa sécurité, le tNETacle utilise le principe de la séparation de privilège: un processus possède des droits de niveau administrateur et un autre non. Le premier ne s'occupe que des actions nécessitant un accés privilégié tel que la création d'un tunnel réseau, alors que le second s'occupe de toute la logique du programme. Cette organisation permet de réduire les dégats occasionnés au système d'exploitation en cas de bug ou de faille de sécurité critique.

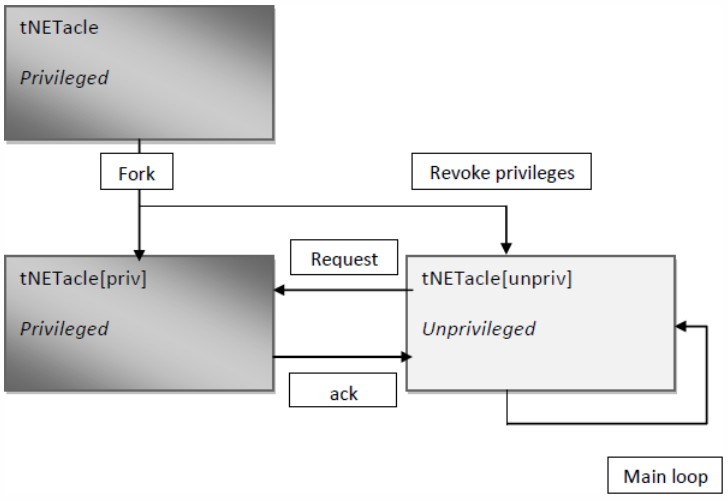

Figure 1: Séparation de privilège au sein du tNETacle

La communication entre ces deux processus se fait via le framework imsg, créé par le projet OpenBSD. Il permet la création et l'émission de messages possédant une en-tête particulière ainsi qu'une variable (généralement un descripteur de fichier).

Ces en-tête sont actuellement: *Interface*

```
enum imsg_type {
        IMSG_NONE,
        IMSG_CREATE_DEV,
        IMSG_SET_IP
};
```

`IMSG_NONE` n'est actuellement pas utilisé, il n'est là que pour des raisons de debug. `IMSG_CREATE_DEV` indique une requette pour la création d'un driver virtuel. `IMSG_SET_IP` permet de configurer l'adresse IP d'un driver virtuel.

La façon dont ces messages sont envoyés n'est pas couverte par ce document, mais par la documentation interne de ce framework.

5.2.2 Pipeline

Un pipeline a été développé pour gérer les différentes étapes d'un paquet réseau avant son envoit à un autre Core. Ce pipeline peut être utiliser dans les deux sens, que se soit pour créer un paquet aussi bien que pour le décoder.

Deux étapes principales de ce pipeline sont le chiffrement et la compression.

5.2.3 Usine à paquet

On peut trouver l'usine à paquet à l'un des bouts du pipeline. Son but principal est de générer des paquets vierges afin de les envoyer dans le pipeline et d'éxtraire les informations proveneant des paquets reçus.

5.2.4 Méta-connexions

Description générale

Les méta-connection (MC) sont une méthode sécurisée et fiable de communication entre diffférents cores. Ces informations regroupent la recherche de clef publique, l'authentification, ... Ces connexions ne transmettent PAS de données génères par des logiciels tiers. L'usage de bande passante y est limité autant que possible pour limiter l'empreinte de tNETacle. En correspondance avec cette faible consommation et le besoin de fiabilité élevée de cette connexion, s'effectue à travers le protocole TCP/IP.

À partir d'ici, le terme MC fait référence aux **`struct mc`** et à l'implémentation des méta-connexions du tNETacle.

Description Technique

Les MC sont en cours de finalisations.

Les MC sont une abstraction construite sur les bufferevents. Leur but est de gérer les différents cas de figure encontrées par ces derniers et de s'assurer que leur composantes et que les connexions sont établies correctement. Veuillez lire la documentation de la libevent sur les bufferevents pour une compréhension plus approfondie.

Structures

Les structures de données figurant les MC sont décritent dans **`struct mc`**. Définie dans `include/mc.h`.

```
struct mc
{
 struct peer {
   struct sockaddr *address;
   int len;
 } p;
 struct bufferevent *bev;
};
```

Les MC sont stockées dans des vecteurs cf **`struct vector`** Définient dans `include/server.h`.

```
struct server {
 /*...*/
 struct vector_mc peers;
 /*...*/
};
```

La **`struct vector_mc`** est un type définit par le header `vector.h`. Se référer à la documentation page 10.

Fonctions

`include/mc.h`

```
void mc_init(struct mc *, struct sockaddr *, int len, struct bufferevent *bev);
void mc_close(struct mc *);
```

Les fonctions `mc_init()` et `mc_close()` peuvent être utilisées pour initialiser et fermer une connexion. Ces deux fonctions n'allouent ni ne libèrent de mémoire. Elles sont appelées pour construire et détruire les **`struct mc`**.

Pour `mc_init`, le paramètre **`struct bufferevent *`**, DOIT être un bufferevent correctement alloué.

6 Client

6.1 Utilité

Le client permet de faire la liaison entre le tNETacle-core et l'utilisateur. Il communiquera avec le tNETacle-core à l'aide d'une connexion réseau, comme le feraient deux tNETacle-core pour communiquer. Dans la suite du document, le client désignera le client graphique. Aucun client textuel (ligne de commande) officiel ne sera fourni.

6.2 Language

Le client est écrit en C++ afin de profiter de la rapidité du langage et de pouvoir avoir accès à la partie objet de celui-ci. Il ne demande pas d'avoir accès à des ressources bas niveau, contrairement au tNETacle-core. De plus, le tNETacle étant écrit en C, nous utiliserons le C++ afin de rester cohérent dans la programmation.

Nous utilisons la bibliothèque Qt pour la partie graphique. Elle est utilisable facilement sur tous les systèmes d'exploitations et est richement fournie en documentation.

6.3 Architecture

L'architecture du client est encore sujette à discutions.

7 Bogues

7.1 Contributions

Veuillez signaler tout bogue rencontré via un ticket sur notre interface dédiée: http://trac.medu.se. Le ticket doit être aussi précis que possible et inclure le numéro de version du logiciel utilisé ainsi que le contexte d'utilisation. Si votre ticket est accepté, il sera assigné à un développeur qui sere responsable de trouver une solution. Vous pouvez également soumettre votre solution en attachant un fichier au ticket. Le patch cera audité par les développeurs et intégré au projet s'il respecte les rêgles établies.

7.2 Problèmes connus

La liste des problèmes connus est établie dans le tableau suivant

Numéro	Emplacement	Description	Résolution
01	libtuntap	Impossible de créer une interface	Oui
02	libtuntap	Mauvais comportement lors de la définition d'une adresse MAC	Oui
03	libtuntap	Imposible d'assigner une adresse MAC sous OpenBSD	Oui
04	libtuntap	Impossible d'assigner un MTU sans détruire le device	Oui
05	libtuntap	Impossible de définir une IPv6	Non