Application du protocole SVC  
dans la communication d’un réseau  
de transport public

Préface

Dans le cadre d’un travail au Groupe de Recherche ERISCS, on dirige un projet pour la RTM concernant un système de communication sécurisée entre les bus et le serveur de surveillance.

Ce document détaille la conception d’un protocole de communication s’appelant SVC, ce que l’on croit, après une petite modification, s’adapte le mieux au besoin de notre collaborateur.

Je me permets de remercier Monsieur T. Muntean, Monsieur A. Février, l’équipe RTM et tous mes collègues d’avoir m’aidé dans ce travail.

Marseille, le 04/09/2016

Xuan Thong DANG

Table de contenu

[I. Problématique 3](#_Toc460927925)

[II. Analyse 4](#_Toc460927926)

[1. La nature de la communication entre les bus et le serveur 4](#_Toc460927927)

[2. Les exigences d’un protocole de communication spécifique 5](#_Toc460927928)

[3. Les solutions existants et la motivation d’un nouveau protocole 6](#_Toc460927929)

[4. Les désavantages de la version original du SVC 7](#_Toc460927930)

[III. Solution 8](#_Toc460927931)

[1. La version modifiée du SVC 8](#_Toc460927932)

[2. L’architecture service-applications 9](#_Toc460927933)

[3. La structure des trames SVC 12](#_Toc460927934)

[4. Le détail d’un processus d’initiation de connexion 14](#_Toc460927935)

[IV. Conclusion 16](#_Toc460927936)

# Problématique

La Régie des Transports Marseillais (RTM) est la société qui est en charge du service de transportation à Marseille – une de trois plus grandes villes de la France. Dans les efforts d’évolution et d’amélioration de ses services, en particulier le réseau de transport en bus, la RTM a besoin d’y mettre en place un système de surveillance, qui est une des exigences de la spécification EBSF.

Pour supporter la communication entre les bus et les serveurs, des réseaux Internet hétérogènes seront utilisés (wifi, 3G, 4G), sur lesquels s’appliquent un protocole de communication sécurisé. Tels protocoles existent sur le marché, mais rien n’a été conçu pour ce problème spécifique.

Notre mission est donc d’analyser la nature de la communication, les avantages et désavantages des solutions existant et de donner la solution la plus adaptée à ce besoin.

# Analyse

## La nature de la communication entre les bus et le serveur

La communication dans le réseau de transport est un type de communication spécifique. Elle subit à des contraintes et obligations faites de la nature du service.

* **Basculement rapide** **de connexion** : en raison du mouvement des bus, la connexion vers le serveur est souvent interrompue quand le bus sort d’une zone de couverture d’une antenne vers celle d’une autre, ou empêchée par de grands obstacles physiques. Le changement du type de réseaux peut aussi se produire au cas où le support du réseau courant ne peut pas desservir toutes les stations qui sont au tour d’un point d’accès.
* **Gros volume de données** : les données récoltées pendant les trajets du bus, concernant le statut de tous les équipements, la position cartographique du véhicule et les points d’accès, et notamment le vidéo-surveillance, sont montées et envoyées vers le serveur. Non seulement un réseau de haut débit mais aussi un bon protocole de transmission qui peut réduire la surcharge de bande passante est requis.
* **Diversité de types et de priorités de données :** les données montées viennent de sources différentes, certaines sont capables de tolérer des erreurs (par exemple quelques trames de vidéo-surveillance), autres requièrent strictement être livrées. On parle aussi des priorités d’envoi, sur lesquelles les données sont traitées et envoyées dans le cas de congestion.
* **Temps réel :** toutes les données de surveillance doivent être transmises au serveur dès qu’elles sont disponibles, pour que les contrôleurs aient des réactions à temps en cas d’incidence.
* **Sécurisée :** les informations collectées dans le bus, y compris les images des passagers sont liées à la vie privée, doivent être transmises et exploitées de façon sécurisée. Une méthode de chiffrement est indispensable dans l’ensemble de solution proposée.

## Les exigences d’un protocole de communication spécifique

Suite aux natures de la communication citées au-dessus, un protocole qui s’adapte à notre besoin doit posséder des propriétés/technologies suivantes:

* Sans connexion : une communication avec un basculement rapide de connexion ne devrait pas utiliser un protocole de transport en mode connecté comme TCP, ce qui augmentera la latence de connexion à cause de sa négociation et des contrôles inutiles. Pour le moment, UDP restera dans notre choix, mais un protocole « hybride » a été considéré comme remplaçant, avec des contrôles simples pour la garantie de livraison de données.
* Reprise de connexion rapide : dans le même but de baisser le temps de conn-exion/reconnexion, on utilisera une couple **sessionID || endPointID** pour distinguer toutes les connexions vers/depuis des hôtes. Une identité de session (sessionID) nous permet d’identifier l’hôte, quand une identité d’extrémité détermine le bout de communication (une instance de l’application). À tout moment, cette couple d’information nous permet de savoir à quelle application appartient un paquet.
* Confidentialité persistante parfaite : pour bien protéger les identités des clients et les données des services, le protocole sécurisé doit supporter la **confidentialité persistante parfaite** (« perfect forward secrecy » en anglais). Comme ça, il garantit que la découverte par un adversaire de la clé privée d'un correspondant (secret à long terme) ne compromet pas la confidentialité des communications passées.
* Courbe elliptique : la courbe elliptique sera utilisée pour réduire la taille des clés en gardant le même niveau de sécurité, économiser la bande passante.
* Indépendance de l’environnement : le protocole est tenté à utiliser sur plusieurs plateformes et appareils, il faudrait donc de ne pas dépendre aux versions du système d’exploitation et des bibliothèques externes, mais à fournir son propre implémentations des algorithmes et services.

## Les solutions existants et la motivation d’un nouveau protocole

Les protocoles sécurisés qu’on se voit de nos jours sont basées essentiellement au TCP : TLS, SSH, OpenVPN, etc…

Déjà, le TCP n’est pas un choix idéal pour notre solution. Il ne satisfait pas la première caractéristique de la connexion, tant que l’établissement et la reprise de la connexion TCP prend toujours du temps considérable. De plus, la garantie de livraison de paquet n’est pas nécessaire en permanence, car dans plupart de temps, les données sont tolérantes aux pertes. La compromise entre la fiabilité et la rapidité du TCP augmente la latence de la connexion, le rend moins efficace de l’utiliser dans un contexte temps réel.

On peut par contre trouver quelques versions UDP de ces protocoles ci-dessus (comme DTLS), ou ces protocoles offrent une option UDP eux-mêmes. Pour garder la même opérabilité qu’en TCP, des modifications supplémentaires sont introduites. Dans l’optique de minimiser la complexité, ce n’est pas non plus une bonne décision.

D’ailleurs, comme indiqué dans la recherche de G. Risterucci et al. [[1]](#footnote-1), les protocoles mentionnés s’imposent encore des limites contre la protection d’identité de client ou l’exposition des informations sensibles.

Dans cette vision née une idée de SVC, un nouveau protocole sécurisé qui surmonte ces limites en proposant une négociation simple et un découplage entre l’authentification et le chiffrement. SVC implémente les algorithmes de chiffrement les plus récents sans utiliser des sources externes, cela facilitera le processus d’administration et maintenance.

Et pourtant, la version origine du SVC a besoin de quelques améliorations pour mieux s’adapter au problème posé. Dans la section suivante, on propose une version légèrement modifiée du SVC avec la conception de l’ensemble de la solution.

## Les désavantages de la version original du SVC

On a trouvé beaucoup d’idées innovantes dans la conception du SVC. Parmi lesquelles, on a choisi à développer l’idée de découplage l’authentification et l’autorisation, et l’idée de protection d’identité du client. Et pourtant, SVC présente dans la conception des points faibles à améliorer.

Dans la version originale proposée par ses auteurs, le protocole commence par une requête depuis le serveur vers le client (après un succès de connexion). Cette approche est justifiée comme une façon de réduire la duration de la négociation. Nous, on n’est pas d’accord avec cette explication. À compter la phase d’initialisation de connexion, SVC consommes 4 exchanges de message.

Le problème le plus grave c’est le fait d’avoir l’étape d’échange de clé commence par le serveur, rend le protocole sensible aux attaques par rejeu. La seule façon pour l’éviter, c’est de rajouter du timestamps, qui ne sera pas une bonne idée.

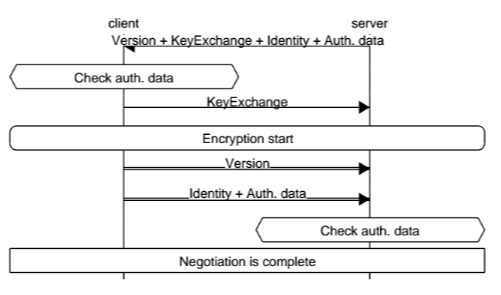


Figure 3.1 – Echange de messages dans la version originale du SVC

# Solution

## La version modifiée du SVC

Avec seulement 3 échanges, on propose une nouvelle approche basé sur le principe du « serrement en 3 phases » qui garde encore les idées basiques du SVC.

Dans le diagramme suivant, on distingue les données « dans les crochets », avec celles qui sont en dehors. Les dernières sont des données de la négociation DH-STS, qui assurent l’autorisation du service, deviennent optionnelles en raison du découplage de service (à expliquer dans la section suivante). Les premières sont des données d’authentification, dépendantes de l’application qui utilise SVC.

Dans la vérification de l’authenticité, l’approche « challenge – proof » est introduite pour éviter tout type d’attaque par rejeu. Le DH-STS est forcément imperméable, seulement vulnérable à quelques attaques du type « unknown key-share »[[2]](#footnote-2).

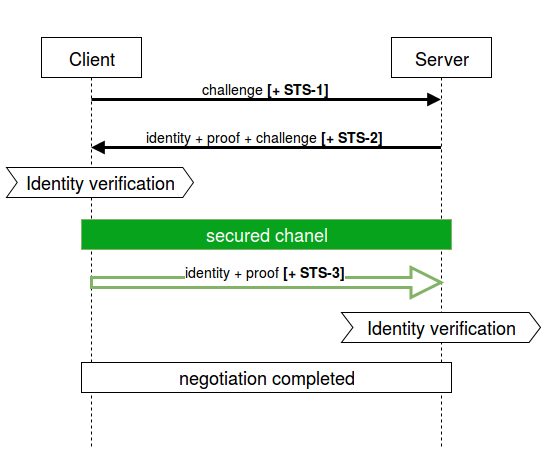


Figure 3.2 – Echange de messages dans la version modifiée du SVC

À comparer avec la version originale, la nouvelle montre des avantages supplémentaires:

* Moins de nombre des échanges (3 contre 4)
* Résistant à l’attaque par rejeu
* Découplage de service

## L’architecture service-applications

À la naissance, SVC est vise à servir une communication interne, c’est-à-dire une communication entre les instances d’une même application. Dans l’environnement multitâche, cette conception montre des défauts:

* chaque connexion doit manager ses propres (mais avec une même politique) paramètres de sécurité, chiffrement. Si on a plusieurs connexions vers une même hôte, toutes les étapes de négociation se refont.
* La mise à jour des protocoles de chiffrement du SVC impose la recompilation de toutes les applications qui l’utilisent.

L’idée est de découper la couche « application », qui est en charge de l’authentification et la communication de données, avec une seule instance de la couche « daemon », qui s’occupe des services de chiffrement et de négociation. Comme ça, on remédie tout de suite les problèmes posés :

* Les services managent toutes les connexions depuis et vers des hôtes. Quand une nouvelle connexion vers une même hôte est détectée, on utilise le service correspondant pour profite d’un canal déjà sécurisé, laisse passer les échanges de clés.
* La mise à jour se fait en simplement replaçant et redémarrant l’instance du daemon. Toutes les applications restent intouchées.

Dans le diagramme ci-dessous se trouvera l’architecture de la solution :

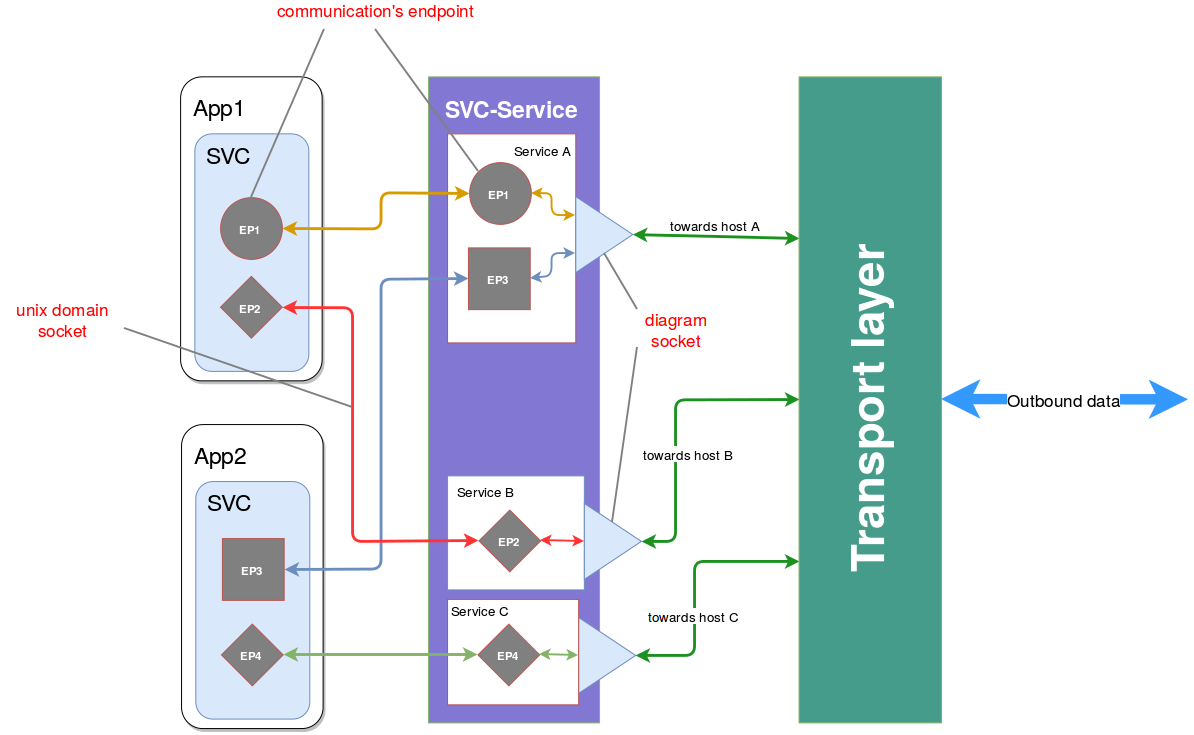


Figure 3.3 – L’architecture service-applications du SVC



Figure 3.4 – Le diagramme de classe côté application



Figure 3.5 – Le diagramme classe côté service

## La structure des trames SVC

Une trame SVC est encapsulée dans la partie de données d’une trame UDP. Il y a deux types de trames SVC : les trames de commandes servent à établir et gérer la connexion, les trames de données quant à elles contiennent les données chiffrées des applications. Une trame du type « données » est toujours chiffrée. Le format de données dans ce type de trame est géré par l’application.

Quel que soit le type de trame, toutes les trames SVC commencent par :

* **Identité de session** – 4 octets: identité unique de la session. Une session est une connexion établie entre deux services (ou bien deux hôtes), qui partage les mêmes paramètres de chiffrement avec tous les chemins de connexion.
* **Identité d’extrémité** – 8 octets : identité d’une extrémité de connexion. Un service SVC utilise cette identité pour distinguer les paquets viennent d’une même hôte. Un chemin de connexion est ce qui connecte 2 extrémités de communication, et ces extrémités partagent une identité identique.

L’utilisation de l’identité de session et d’extrémité permet la reprise rapide de connexion en cas de reconnexion.

Après l’identité de session et d’extrémité, succède un octet qui détermine le type de la trame et contient des paramètres pour la gestion de données.

* Données/commande: 7th bit, égale 1 si la trame est du type commande, 0 si la trame contient des données
* Réponse depuis daemon: 6th bit, égale 1 si la trame reçue est une notification du daemon local
* Pas d’usage pour le moment : 5th et 4th bits
* Chiffré: 3th bit, égale 1 si la partie de données de la trame est chiffrée
* TCP utilisé: 2nd bit, égale 1 si cette trame requit une garantie de livraison, à lire et assurer par la couche de transport
* Priorité : 1er et 0th bits, la priorité est parmi URGENT, HIGH, NORMAL, LOW, qui sera traitée par la couche transport

Une visualisation d’une trame SVC est démontré comme ci-dessous :

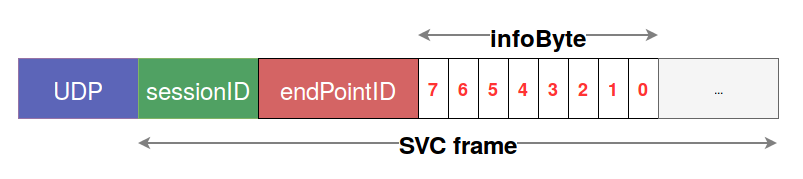
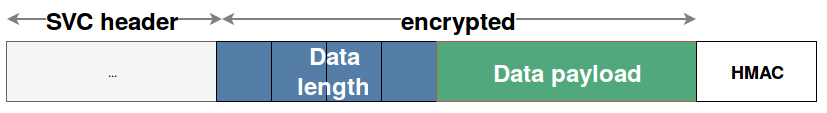


Figure 3.4 – La structure générale d’une trame SVC

Une trame de données du SVC contient, après l’entête, 4 octets de la longueur de la charge utile de données, suivie par la charge elle-même et le HMAC (l’algorithme et la longueur du HMAC dépend de la version du SVC). Une trame de données doit être toujours chiffrée.

Figure 3.5 – Une trame de données du SVC

Une trame de commande du SVC commence par l’identité de commande (CID), suivie par le nombre de paramètres (PRC). Le reste est la partie des paramètres, dans laquelle chaque paramètre est précédé par 2 octets de longueur. Sauf les commandes d’initialisation de connexion qui ne sont pas chiffrées, toutes les commandes restant sont chiffrées, et se terminent par un HMAC.

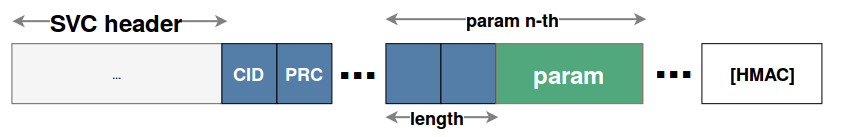


Figure 3.6 – Une trame de commande du SVC

## Le détail d’un processus d’initiation de connexion

Une application crée une instance de SVC et demande la connexion. Dans la demande contient l’adresse de l’hôte à distance, qui sera utilisée par le « daemon » pour décider s’il doit créer un nouveau service (chaque service est en charge de connexion à un hôte unique).

À noter qu’il y a une seule instance du « daemon » qui s’exécute dans le background et qui est en charge de toutes instances du service. Pour pouvoir distinguer les demandes de connexion des applications différentes, le STEP\_1 doit lui communique une identité de l’application (appID). Cette identité est liée à l’application et n’est pas au client sur lequel l’application s’exécute.



Figure 3.7 – Le processus d’initialisation de connexion du SVC

Une nouvelle identité de session est communiquée (si besoin) dans le STEP\_3 pour identifier la session entre deux hôtes et qui servira à la reprise rapide de session.

Une demande de connexion et ses extrémités seront détruites si elle n’est pas réussie après un certain de temps.

# Conclusion

La phase de conception de cette solution a quasiment fini. Pendant la phase d’implémentation, des petites modifications peuvent être introduites, mais toujours en respectant les principes déterminés.

Une partie de l’implémentation est déjà en cours d’être réalisée. Dès une version complète est disponible, on va tester et comparer sa performance avec les protocoles existant pour pouvoir l’améliorer. Une conception du HTP (Hibrid Transmission Protocol) et son implémentation est aussi un de nos travaux au futur proche pour supporter le SVC.

1. G. Risterucci, T. Muntean, L. Mugwaneza. *A new Secure Virtual Connector approach for communication within large distributed systems*. ERISCS Research Group. [↑](#footnote-ref-1)
2. Blake-Wilson, S.; Menezes, A. (1999), "Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol", *Public Key Cryptography*, Lecture Notes in Computer Science, **1560**, Springer, pp. 154–170 [↑](#footnote-ref-2)