Une approche de génération automatique de configuration basée sur les modèles pour les réseaux TSN

Maxime Samson*†, Thomas Vergnaud*, Éric Dujardin*, Laurent Ciarletta† and Ye-Qiong Song†
* Thales Research & Technology – Palaiseau, France
{maxime.samson – eric.dujardin – thomas.vergnaud}@thalesgroup.com
† LORIA – Université de Lorraine – Nancy, France
{maxime.samson – ye-qiong.song – laurent.ciarletta}@loria.fr

***Abstract*—Les nouvelles fonctionnalités qu'apportent les standards définis par le groupe de travail IEEE 802.1 TSN à la commutation Ethernet permettent son utilisation pour des réseaux temps réel. Ces nouvelles fonctionnalités rendent possible la conception de réseaux Ethernet déterministes, mais au prix d'un effort de configuration très important. Cette difficulté de configuration s'applique à la fois aux équipements réseaux et aux outils utilisés pour concevoir ces réseaux, par exemple des simulateurs.**

Dans cet article, nous présentons une approche basée sur les modèles qui permet la génération automatique de la configuration d'un réseau TSN pour des outils tels que des simulateurs. Cette approche simplifie l'étape de configuration réseau et assure la cohérence entre les configurations générées pour les différentes cibles.

I. Introduction

Time Sensitive Networking (TSN) est un ensemble de standards développés par le groupe de travail IEEE 802.1 TSN, qui était anciennement nommé Audio/Video Bridging (AVB).

Cet ensemble de standards a pour objectif de compléter le standard 802.1Q [1] afin de rendre possible l'utilisation d'Ethernet pour des communications déterministes.

Pour ce faire, les standards TSN apportent un ensemble de nouvelles fonctionnalités, p.ex. une notion de temps commune au réseau établie par synchronisation temporelle, la réservation des ressources pour les flux de données, la régulation des flux.

Un réseau Ethernet déterministe permet des communications à criticité mixte avec des garanties de qualité de service et une bande passante élevée, ce qui intéresse des secteurs comme l'automatisation industrielle, l'automobile et l'aviation.

Dans le secteur automobile par exemple, les réseaux TSN présentent plusieurs avantages. TSN permet des communications de criticité différentes. Il est donc possible de n'utiliser qu'un seul réseau pour les communications critiques, qui utilisent généralement un bus Controller Area Network (CAN), et un autre réseau (p.ex. MOST) pour les communications audio et vidéo liées au divertissement [2]. L'utilisation d'un réseau unique permet notamment une réduction du câblage et donc du poids des véhicules.

Les nouvelles fonctionnalités apportées par les standards TSN entraînent une augmentation de la complexité de la configuration qu'il faut déployer sur le réseau [3], [4].

Cela s'applique également à l'utilisation de simulateurs ou d'émulateurs réseau, ce qui est courant lors de la phase de conception. Ce problème vient s'ajouter à un problème existant lié à l'utilisation de ces outils : la nécessité de savoir configurer chacun des outils que l'on souhaite utiliser.

L'approche basée sur la génération automatique de configuration à partir de modèles que nous présentons dans cet article permet de résoudre ces problèmes. La génération de la configuration permet tout d'abord d'éviter les erreurs humaines. Elle permet également de ne pas être obligé de savoir configurer chacun des outils utilisés lors de la phase de conception. Cette approche permet donc un gain de temps important.

Des travaux sur la génération automatique de configuration pour les réseaux TSN existent déjà. Dans [5], TSNSched, un outil de génération de configuration pour un mécanisme de régulation de flux de TSN, le Time Aware Shaper (TAS), est présenté. Cet outil utilise un démonstrateur automatique et un système de contraintes. L'approche de modélisation de réseaux qui est présentée n'est pas découplée de l'outil de génération, ce qui ne la rend pas exploitable par d'autres outils. La limite principale de ces travaux est que la configuration générée ne concerne que le TAS est n'est donc pas exploitable par un outil de conception comme un simulateur.

L'outil présenté dans [4] permet de générer automatiquement la configuration d'un réseau TSN pour un simulateur réseau. Cet outil prend la forme d'une extension du simulateur et lui est donc liée. Il n'est donc pas possible d'utiliser cet outil pour générer de la configuration pour différents outils de conception ou pour du matériel réel.

Dans la suite de cet article, nous commencerons par expliquer plus en détail le problème posé par la complexité de la configuration d'un réseau TSN puis nous présenterons la solution que nous proposons et son implémentation.

II. TSN

Les standards TSN définissent plusieurs mécanismes de régulation de flux, dont certains sont basés sur la division temporelle, qui permettent de garantir le respect des contraintes temps réel des flux de données. La synchronisation temporelle de tous les nœuds du réseau est un mécanisme de base de TSN. Cette synchronisation est assurée par le standard IEEE 802.1AS qui définit le Generic Precision Time Protocol (gPTP).

Afin d'assurer que le réseau sera bien capable de respecter les contraintes temps réel des différents flux de données, les ressources nécessaires doivent être réservées au niveau des commutateurs. Par exemple, la somme des bandes passantes nécessaires aux flux transmis par un port d'un commutateur doit être inférieure à la capacité totale de ce port.

Un des mécanismes de régulation de flux de TSN est le Credit-Based Shaper (CBS). Il lisse le trafic, ce qui empêche des gros flux de données de saturer temporairement le réseau. Pour cela, il définit des classes de trafic, et l'utilisateur leur attribue le taux auquel elles gagneront des crédits. Les trames appartenant à ces classes de trafic ne sont transmises que si la quantité de crédits de leur classe est positive ou nulle. Quand des trames sont émises, leur classe de trafic perd des crédits, et elle en gagne pendant qu'elles sont en attente de transmission.

L'utilisation du CBS implique nécessairement des périodes pendant lesquelles les trames des classes de trafic qu'il régule seront en attente. Ce comportement n'est pas souhaitable pour la régulation des flux de données les plus critiques, pour lesquels la latence doit être la plus basse possible. Pour atteindre cet objectif, le standard IEEE 802.1Qbv définit le Time Aware Shaper (TAS). Le rôle de ce mécanisme est de répartir l'accès aux liens du réseau dans le temps afin d'isoler les flux les plus critiques et de leur garantir une latence et une gigue minimale. Le TAS définit, au niveau des ports des commutateurs, la durée d'un cycle puis divise ce cycle en plusieurs fenêtres pendant lesquelles seules certaines classes de trafic pourront transmettre. Cette division temporelle est présentée dans la figure 1 sous la forme d'une table qui contient trois fenêtres temporelles. La sélection finale de la prochaine trame à transmettre est faite en respectant le code de priorité, présent pour chaque trame dans l'entête défini par le standard IEEE 802.1Q.

La figure 1 présente les mécanismes de régulation de flux utilisés par les ports de sortie des commutateurs d'un réseau TSN. On voit que les nœuds émetteurs des flux de données suivent chacun leur ordonnancement; en fait, chaque port de sortie des commutateurs peut être configuré de façon particulière.

Chaque port de chaque commutateur susceptible de transmettre un flux de données doit être configuré individuellement et ces configurations doivent permettre de respecter les contraintes de chaque flux. Cette multiplicité des configurations est ce qui crée la complexité de la configuration globale d'un réseau TSN, qui vient s'ajouter aux problématiques classiques, comme le routage.

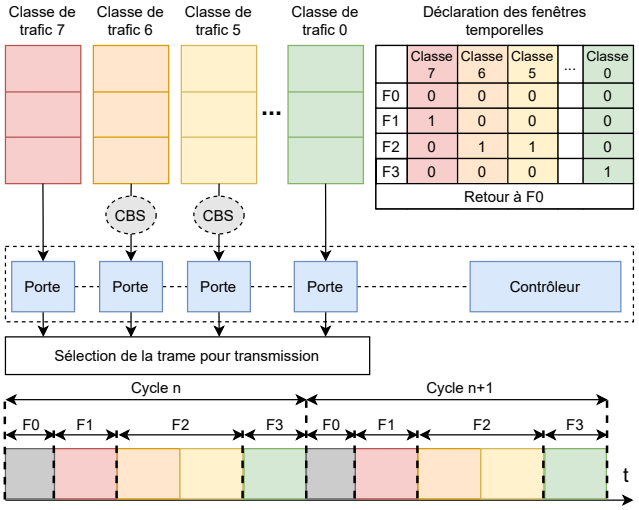

Fig. 1. Représentation des mécanismes utilisés par un port de sortie d'un commutateur TSN

III. Solution

A. *Présentation de la solution*

Afin de résoudre les problèmes que pose la complexité de la configuration d'un réseau TSN, notre approche comporte plusieurs étapes, présentées dans la figure 2.

La première étape est l'expression des contraintes que le réseau doit être capable de respecter. Ces contraintes ne suivent pas de formalisme particulier et peuvent concerner plusieurs aspects du réseau.

Les contraintes les plus importantes sont celles qui concernent les différents flux de données qui doivent circuler sur le réseau. Elles doivent tout d'abord permettre de dimensionner les flux. Par exemple, dans le cas de flux périodiques, elles expriment la taille des données et la période de transmission. Elles doivent ensuite définir la latence que le flux ne doit jamais dépasser.

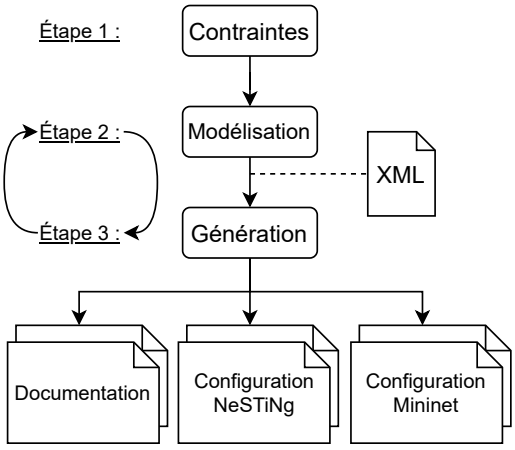

Fig. 2. Présentation de l'approche

La topologie du réseau peut également être contrainte. Le réseau peut être soumis à des problématiques d'espace ou de poids comme dans une voiture ou sur un drone. Les équipements peuvent en être une autre source, par exemple le nombre de commutateurs à utiliser ou le nombre de ports que possède chacun d'entre eux.

La deuxième étape de notre approche est la modélisation. L'objectif de cette étape est de créer une représentation de chaque élément du réseau qui respecte les contraintes exprimées à l'étape précédente et qui contienne les informations nécessaires à la génération automatique de configuration. Le modèle ainsi créé servira ensuite à assurer la cohérence entre les différentes configurations générées.

Enfin, la dernière étape est la génération de la configuration. Cette étape se base sur le modèle créé à l'étape précédente et la configuration est générée pour les cibles spécifiées par l'utilisateur.

Pour assister la conception du réseau, il est possible d'itérer sur les étapes 2 et 3 en évaluant le comportement du réseau dans un simulateur pour faire évoluer le modèle du réseau, et générer à nouveau une configuration.

B. Mise en œuvre de la solution

1) Modélisation: Notre modèle de déploiement est inspiré des concepts du standard MARTE [6], et plus particulièrement du chapitre GRM – Generic Resource Modeling. Nous nous reposons sur deux concepts qui y sont définis: ComputingResource et CommunicationMedia.

Nous reprenons le concept de ComputingResource pour décrire un nœud de calcul au sens large. Dans notre cas, nous l'utilisons pour représenter une machine ou un commutateur TSN. À partir de CommunicationMedia, nous décrivons les bus au sens large. Dans notre cas, nous l'utilisons pour modéliser les liens Ethernet et les flux.

Nous n'utilisons donc pas l'intégralité de ce qu'offre le standard MARTE mais nous nous en inspirons en l'adaptant. MARTE est un standard bien connu du domaine des systèmes embarqués temps réel, ce statut fait de lui un bon point de départ pour la modélisation de réseaux Ethernet déterministes.

Notre approche repose sur la notion de *ressource*: nous utilisons les termes ComputationResource (compRsc) et CommunicationResource (commRsc).

Nous faisons la distinction entre définition (compRscDef et commRscDef) et instanciation (compRsc et commRsc). La définition d'un commutateur permet d'expliciter l'ensemble des paramètres nécessaire à sa caractérisation. L'instanciation d'un commutateur consiste à créer un exemplaire de commutateur faisant référence à la définition; les valeurs des paramètres sont spécifiées dans l'instanciation. Le listing 1 montre la définition d'un flux de données.

```
<commRscDef name="Stream">
  <configParam max="1" min="0" name="deadline"
               type="time_t"/>
  <configParam max="1" min="1" name="payload"
               type="payload_t"/>
  <configParam max="1" min="1" name="pcp"
               type="pcp_t"/>
  <configParam max="1" min="1" name="vlan_id"
               type="vlan_id_t"/>
  <rscParam max="1" min="1" name="talker">
    <allowedRscDef ref="Ethernet_Interface"/>
  </rscParam>
  <structParam max="-1" min="1"
               name="listeners">
    <rscParam max="1" min="1"
              name="listener_port">
      <allowedRscDef
              ref="Ethernet_Interface"/>
    </rscParam>
    <structParam max="-1" min="1"
                 name="paths">
      <rscParam max="-1" min="1"
                name="path_member">
        <allowedRscDef
                ref="Ethernet_Interface"/>
      </rscParam>
    </structParam>
  </structParam>
</commRscDef>
```

Listing 1. Définition d'un flux de données

Le listing 2 montre l'instanciation d'un flux de données conforme à cette définition. Ce flux de données appartient à la classe de trafic la plus prioritaire et son délai de bout en bout ne doit pas dépasser 2 millisecondes. C'est un flux périodique dont les trames sont émisent toutes les 250 microsecondes et contiennent 128 octets. Il est émis par *talker1*, a pour seul destinataire *listener1* et doit traverser *switch1* et *switch2*.

```
<commRsc def="Periodic_Stream" name="stream1">
  <config def="deadline" value="{2, ms}"/>
  <config def="payload" value="{128, B}"/>
  <config def="pcp" value="7"/>
  <config def="period" value="{250, us}"/>
  <config def="vlan_id" value="1"/>
  <rscConfig def="talker"
             value="talker1_eth0"/>
  <structConfig def="listeners">
    <rscConfig def="listener_port"
               value="listener1_eth0"/>
    <structConfig def="paths">
      <rscConfig def="path_member"
                 value="switch1_eth0"/>
      <rscConfig def="path_member"
                 value="switch2_eth2"/>
    </structConfig>
  </structConfig>
</commRsc>
```

Listing 2. Instanciation d'un flux de données

Notre travail a consisté à établir les définitions des concepts, tels le commutateur ou le flux de donnée. Le travail de l'utilisateur consiste alors à instancier ces concepts. Il sait ainsi quelles informations il doit fournir pour chaque instanciation, ce qui le guide dans la spécification du réseau.

2) Génération: Notre outil de génération de configuration s'appuie sur les modèles d'instanciation établis au format XML lors de l'étape précédente. L'utilisateur spécifie le modèle à utiliser et les cibles pour lesquelles la configuration doit être générée.

Après avoir extrait les données du modèle, l'outil génère un ensemble de fichiers contenant de la documentation sur le modèle utilisé et les fichiers de configurations. Les cibles pour lesquelles la configuration peut être générée sont, pour l'instant, Mininet [7] et NeSTiNg [8].

Mininet est un émulateur réseau conçu pour SDN (Software Defined Network) permettant de créer un réseau composé de commutateurs, de liens et de terminaux virtuels. Un réseau virtuel permet, sur une seule machine, de prototyper, de tester et de déboguer un réseau avant de le déployer. Il est possible de spécifier la topologie à utiliser grâce à une interface de programmation Python. Cette interface permet de facilement effectuer des modifications sur la topologie et donc de tester différentes configuration du réseau.

NeSTiNg est un modèle de simulation de réseau TSN qui repose sur OMNeT++[1] et son framework INET[2]. Il met à disposition de ses utilisateurs des éléments permettant de simuler des réseaux TSN dans OMNeT++. Ces éléments incluent des commutateurs et des terminaux qui possèdent les fonctionnalités spécifiques à TSN.

La configuration d'une simulation utilisant NeSTiNg se fait par le biais de plusieurs fichiers : un fichier contenant la description de la topologie, un fichier contenant les paramètres de la simulation et plusieurs fichiers XML décrivant les flux de données, le routage et la configuration du TAS. Ces fichiers suivent tous une syntaxe stricte et peuvent donc être générés automatiquement.

La génération de la topologie dans les deux formats (Mininet et NeSTiNg) à partir d'un unique modèle permet de conserver la cohérence de la configuration.

Le listing 3 montre l'initialisation d'un flux de données dans NeSTiNg.

```
talker1.numApps = 1
talker1.app[0].typename =
    ↪ "UdpScheduledTrafficApp"
talker1.app[0].trafficGenerator.localPort =
    ↪ 1000
talker1.app[0].scheduleManager.
    ↪ initialAdminSchedule = xmldoc("xml/flows
    ↪ .xml", "/schedules/datagramSchedule[@id
    ↪ ='0']")

listener1.numApps = 1
listener1.app[0].typename = "UdpSink"
listener1.app[0].localPort = 1000
```

Listing 3. Initialisation d'un flux de données dans NeSTiNg

Le listing 4 montre l'instanciation d'un flux de données dans NeSTiNg à laquelle l'initialisation fait référence.

```
<datagramSchedule id="0" cycleTime="250us">
  <event payloadSize="128B"
         destAddress="listener1"
         destPort="1000" pcp="7" vid="1"/>
</datagramSchedule>
```

Listing 4. Instanciation d'un flux de données dans NeSTiNg

[1]https://omnetpp.org/
[2]https://inet.omnetpp.org/

Pour le modèle d'un réseau composé de 3 commutateurs et de 6 terminaux, l'outil de génération de configuration utilisé en spécifiant NeSTiNg comme cible génère environ 650 lignes dont environ 400 lignes de configuration pour NeSTiNg. Le temps d'exécution de la génération est en moyenne, sur 20 exécutions, de 123 ms pour une machine sous openSUSE Leap 15.2 équipée d'un Core i7 6820HQ.

IV. Conclusion

Dans cet article, nous avons présenté une approche de génération automatique de configuration basée sur des modèles. Cette approche consiste à réaliser un modèle du réseau pour lequel l'utilisateur veut générer la configuration, en respectant les contraintes définies en amont. Chaque élément d'un réseau, p.ex. les commutateurs ou les terminaux, possède sa propre définition. Le modèle doit respecter ces définitions et contenir les données nécessaires à l'instanciation de chacun de ces éléments. Le modèle s'exprime dans une syntaxe XML simple. Notre générateur de configuration génère alors la configuration pour la cible choisie.

Notre approche présente l'avantage de faciliter l'utilisation de différents outils lors de la phase de conception d'un réseau. Elle évite à l'utilisateur d'avoir à maîtriser la configuration de chacun de ces outils et elle assure la cohérence entre les différentes configurations puisqu'elles ont toutes été générées à partir du même modèle.

Nous avons prévu comme travaux futurs d'améliorer le générateur de configuration. Nous envisageons d'ajouter une fonctionnalité permettant de compléter des modèles. Cela permettra d'alléger la charge de l'utilisateur lors de l'étape de modélisation, notamment en calculant les durées des cycles TAS et des fenêtres temporelles, en s'appuyant, par exemple, sur un outil comme TSNSched [5]. Nous prévoyons également d'ajouter la possibilité de générer la configuration pour du matériel réel, par exemple avec un modèle YANG.

References

[1] *IEEE 802.1Q 2018 – Bridges and Bridged Networks*, IEEE Std., 2018.
[2] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.
[3] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 42–47, 2018.
[4] B. Houtan, A. Bergström, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "An automated configuration framework for tsn networks," in *22nd IEEE International Conference on Industrial Technology (ICIT'21)*, March 2021.
[5] A. C. T. d. Santos, B. Schneider, and V. Nigam, "Tsnsched: Automated schedule generation for time sensitive networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 69–77.
[6] *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, OMG Std., 2019.
[7] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*. ACM Press, pp. 1–6.
[8] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Durr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *2019 International Conference on Networked Systems (NetSys)*. IEEE, pp. 1–8.