HAL
open science

Active self-diagnosis in telecommunication networks

Carole Hounkonnou

To cite this version:

Carole Hounkonnou. Active self-diagnosis in telecommunication networks. Other [cs.OH]. Université de Rennes, 2013. English. NNT : 2013REN1S086 . tel-00931853v2

HAL Id: tel-00931853

https://theses.hal.science/tel-00931853v2

Submitted on 20 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ANNÉE 2013

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole doctorale Matisse

présentée par

Carole Hounkonnou

Préparée à l'unité de recherche INRIA
Institut National de Recherche en Informatique et Automatique
Composante universitaire ISTIC

Active self-diagnosis in telecommunication networks.

Auto-diagnostic actif dans les réseaux de télécommunications.

Thèse soutenue à Rennes le 12 Juillet 2013

devant le jury composé de :

Francine KRIEF
Professeur, IPB, Université de Bordeaux 1 / *rapporteur*

Yacine GHAMRI-DOUDANE
Maître de Conférences (HDR), Université de Paris Est / *rapporteur*

Xavier LAGRANGE
Professeur, Télécom Bretagne / *examinateur*

Christian DESTRÉ
Directeur de projet, Orange Labs / *examinateur*

Laurent CIAVAGLIA
Directeur de projet, Alcatel-Lucent Bell Labs / *examinateur*

Éric FABRE
Directeur de recherche, INRIA / *directeur de thèse*

Acknowledgements

First and above all, I praise God, the almighty for providing me this opportunity and granting me the ability to proceed successfully in spite of many challenges faced.

It would not have been possible to complete this doctoral thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

I would like to express my sincere thanks to *Éric Fabre*, my supervisor, for his warm encouragement, critical comments, and thoughtful guidance throughout the course of this thesis. I also reserve special thanks to *Samir Ghamri-Doudane* for his time, valuable inputs and contributions to this work.

I would also like to thank the members of my thesis committee. I place on record my sincere gratitude to *Francine Krief* and *Yacine Ghamri-Doudane* for spending time reviewing my thesis. I am also grateful to *Xavier Lagrange*, *Christian Destré*, and *Laurent Ciavaglia* for acting as examiners of my thesis. I want to thank you all for letting my defence be an enjoyable moment, and for your brilliant comments and suggestions, thanks to you.

There were many people who gave interesting feedback and valuable suggestions, for which I would like to thank them: my colleagues at INRIA and the members of the *High Manageability* joint research group of Alcatel-Lucent Bell Labs and INRIA. A special thanks to *César Viho* whose advice and support have been invaluable on both an academic and a personal level. I am also very thankful to *Nathalie Bertrand* and *Loïc Hélouët* for helping me practicing my presentation and for providing constructive feedback. All these people have been a source of friendship as well as good advice and collaboration.

Last but not least, a special thought is devoted to my family. Words can not express how grateful I am to my parents, *Anastasie* and *Oussou Hounkonnou*, and my brothers, *Orens*, *Oswald* and *Bernard*, for all of the sacrifices that you've made on my behalf. Your prayers for me was what sustained me thus far. God bless you all.

Abstract

While modern networks and services are continuously growing in scale, complexity and heterogeneity, the management of such systems is reaching the limits of human capabilities. Technically and economically, more automation of the classical management tasks is needed. This has triggered a significant research effort, gathered under the terms self-management and autonomic networking.

The aim of this thesis is to contribute to the realization of some self-management properties in telecommunication networks. We propose an approach to automatize the management of faults, covering the different segments of a network, and the end-to-end services deployed over them. This is a model-based approach addressing the two weaknesses of model-based diagnosis namely: a) how to derive such a model, suited to a given network at a given time, in particular if one wishes to capture several network layers and segments and b) how to reason with a potentially huge model, if one wishes to manage a nation-wide network, for example.

To address the first point, we propose a new concept called self-modeling that formulates off-line generic patterns of the model, and identifies on-line the instances of these patterns that are deployed in the managed network. The second point is addressed by an active self-diagnosis engine, based on a Bayesian network formalism, that consists in reasoning on a progressively growing fragment of the network model, relying on the self-modeling ability: more observations are collected and new tests are performed until the faults are localized with sufficient confidence.

This active diagnosis approach has been experimented to perform cross-layer and cross-segment alarm management on an IP Multimedia Subsystem (IMS) network.

Chapter 1

Résumé en français

Contents

1.1 Contexte 3
1.2 Introduction 4
1.3 Structure des réseaux IMS 6
1.4 L'auto-modélisation: support de la localisation de pannes . 8
1.5 Raisonnement avec les réseaux Bayésiens génériques 13
1.6 Résultats 15
1.7 Conclusion 17

1.1 Contexte

Les réseaux de télécommunications deviennent de plus en plus complexes, notamment de par la multiplicité des technologies mises en œuvre, leur couverture géographique grandissante, la croissance du trafic en quantité et en variété, mais aussi de par l'évolution des services fournis par les opérateurs. Tout ceci contribue à rendre la gestion de ces réseaux de plus en plus lourde, complexe, génératrice d'erreurs et donc coûteuse pour les opérateurs. On place derrière le terme « réseaux autonomiques » l'ensemble des solutions visant à rendre la gestion de ce réseau plus autonome.

L'objectif de cette thèse est de contribuer à la réalisation de certaines fonctions autonomiques dans les réseaux de télécommunications. Nous proposons une stratégie pour automatiser la gestion des pannes tout en couvrant les différents segments du réseau et les services de bout en bout déployés au-dessus. Il s'agit d'une approche basée modèle qui adresse les deux difficultés du diagnostic basé modèle à savoir: a) la façon d'obtenir un tel modèle, adapté à un réseau donné à un moment donné, en particulier si l'on souhaite capturer plusieurs couches et segments réseau et b) comment raisonner sur un modèle potentiellement énorme, si l'on veut gérer un réseau national par exemple.

Pour répondre à la première difficulté, nous proposons un nouveau concept : l'auto-modélisation qui consiste d'abord à construire les différentes familles de modèles génériques, puis à identifier à la volée les instances de ces modèles qui sont déployées dans le réseau géré. La seconde difficulté est adressée grâce à un moteur d'auto-diagnostic actif, basé sur le formalisme des réseaux Bayésiens et qui consiste à raisonner sur un fragment

du modèle du réseau qui est augmenté progressivement en utilisant la capacité d'auto-modélisation: des observations sont collectées et des tests réalisés jusqu'à ce que les fautes soient localisées avec précision.

Cette approche de diagnostic actif a été expérimentée pour réaliser une gestion multi-couches et multi-segments des alarmes dans un réseau IMS (IP Multimedia Subsystem).

1.2 Introduction

Plusieurs approches pour la gestion des pannes utilisent des méthodes de type « boite noire » ou des méthodes d'apprentissage: réseaux de neurones, cartes auto adaptatives, apprentissage statistique (Kavulya et al., 2011), techniques de dictionnaire (Reali and Monacelli, 2009), découverte de chroniques (Dousson, 1996; Dousson and Duong, 1999; Kavulya et al., 2012). De telles stratégies souffrent généralement d'un manque de données classifiées reliant les pannes aux symptômes observés. Ces stratégies résistent mal aux phénomènes de familles de symptômes complexes dus aux fautes multiples, à l'asynchronisme ou à la perte des observations. Par ailleurs, ces méthodes sont difficiles à maintenir lorsque le réseau évolue. De telles méthodes sont donc appropriées pour des corrélations d'alarmes simples, à petite échelle sur des topologies fixes, où la convergence du processus d'apprentissage est assurée.

Pour atteindre les objectifs mentionnés plus haut, nous devons adopter une stratégie basée modèle, qui consiste tout d'abord à construire un modèle du réseau géré, à établir les relations entre le fonctionnement correct ou incorrect et les symptômes ou signaux observés, et ensuite à dériver des algorithmes de gestion d'évènements basés sur ce modèle. C'est la stratégie adoptée par plusieurs contributions (voir les techniques de traversée de modèle dans (Steinder and Sethi, 2004a), et leur généralisation en méthodes théoriques graphiques). D'autres contributions assemblent la connaissance experte à propos des disfonctionnements et de leur conséquences, ou des symptômes et de leur causes, dans des graphes causaux qui forment le support pour un raisonnement automatique (éventuellement distribué) (Lu et al., 2011; Grosclaude, 2008). Une autre approche majeure modélise ou découvre à partir des données, les dépendances (logiques ou statistiques) entre les ressources, les pannes initiales et les symptômes observables, puis, les assemble en réseaux Bayésiens ou en objets similaires (Bouloutas et al., 1994, 1995; Fabre et al., 2004). Les moteurs d'inférence sur de telles structures sont standards et prêts à l'usage. En effet, ce sujet ainsi que la distribution de ces techniques sont des thèmes bien couverts par la littérature (Fabre et al., 2004, 2005).

Cependant, comme en témoigne les contributions ci-dessus, les techniques basées modèle soulèvent toujours deux difficultés majeures: a) comment obtenir un tel modèle adapté à un réseau donné à un moment donné, en particulier si l'on souhaite capturer plusieurs couches et segments réseau, et b) comment raisonner sur un modèle potentiellement énorme, si l'on veut gérer un réseau national par exemple. Cette thèse propose une contribution à ces deux difficultés.

Obtenir un modèle du réseau géré est loin d'être trivial, surtout si l'on souhaite

capturer les propagations de pannes inter-couche et inter-segment. La première source d'information que l'on doit utiliser est bien sûr la topologie du réseau, qui a été considérée dans plusieurs contributions (par exemple (Bouloutas et al., 1994)), y compris les outils professionnels dédiés à des technologies réseau spécifiques. Cependant, l'on devrait aller plus loin et agréger différentes sources d'information, généralement trouvées dans les normes, dans les descriptions de protocoles, et dans la connaissance experte si celle-ci est disponible.

S'appuyant sur l'expérience de notre équipe dans la modélisation des propagations de fautes et d'alarmes (Fabre et al., 2004), nous proposons ici un nouveau concept: l'auto-modélisation. L'auto-modélisation consiste tout d'abord à identifier les différentes familles de ressources réseau qui doivent être gérées, et la façon dont ces ressources sont structurées et liées les unes aux autres, en suivant la construction hiérarchique habituelle des réseaux. Cela donne une collection de modèles génériques, de ressources et de dépendances entre celles-ci, conçus selon une grammaire spécifique. Ensuite, en explorant le réseau géré, on peut alors créer des instances de ces modèles génériques autant de fois que celles-ci sont découvertes dans la topologie du réseau. Comme ces instances partagent certaines ressources, cette construction se traduit par une structure à grande échelle où des tendances similaires sont dupliquées et se chevauchent partiellement. Le modèle obtenu de cette manière correspond parfaitement à un réseau donné et peut être utilisé pour le diagnostic.

En ce qui concerne le moteur de diagnostic, nous proposons de traduire le modèle des ressources réseau et de leur dépendances dans le formalisme des réseaux Bayésiens, qui semble faire l'objet d'un consensus au sein de la communauté de gestion de réseau. Les réseaux Bayésiens peuvent facilement mêler dépendances statistiques et contraintes/logique, tout en permettant un apprentissage statistique limité (indentification des paramètres) quand les données sont disponibles. Ils peuvent aussi s'adapter aux réseaux de systèmes dynamiques (Fabre, 2007; Fabre et al., 2004; Fabre and Hadjicostis, 2006). Le raisonnement probabiliste sur les réseaux Bayésiens est très bien documenté et permet d'associer des observations ou des résultats de test à l'état de variables pertinentes cachées.

Néanmoins, les réseaux Bayésiens s'avèrent inappropriés lorsqu'il s'agit de faire face à des modèles potentiellement énormes. Par conséquent, nous proposons d'adapter le formalisme des réseaux Bayésiens pour a) explorer seulement une partie du modèle, en commençant par les ressources impliquées dans une panne donnée qui doit être expliquée, et b) introduire/révéler progressivement plus de ressources (variables) pour obtenir plus d'observations et réaliser de nouveaux tests, dans le but de localiser l'origine de la panne avec plus de précision.

Pour appuyer ces deux directions de recherche, la thèse considère la gestion des pannes dans les réseaux IMS, tout en capturant plusieurs segments réseau (*access*, *metro* et *core*), et les services de bout en bout déployés au-dessus (Bertin et al., 2007).

1.3 Structure des réseaux IMS

Trois couches multi-résolution

En partant des descriptions classiques des réseaux IMS, l'objectif ici est d'identifier les ressources impliquées dans de tels réseaux, et la façon dont ces ressources sont structurées et dépendent les unes des autres. Cette connaissance sera utilisée pour diagnostiquer les dysfonctionnements.

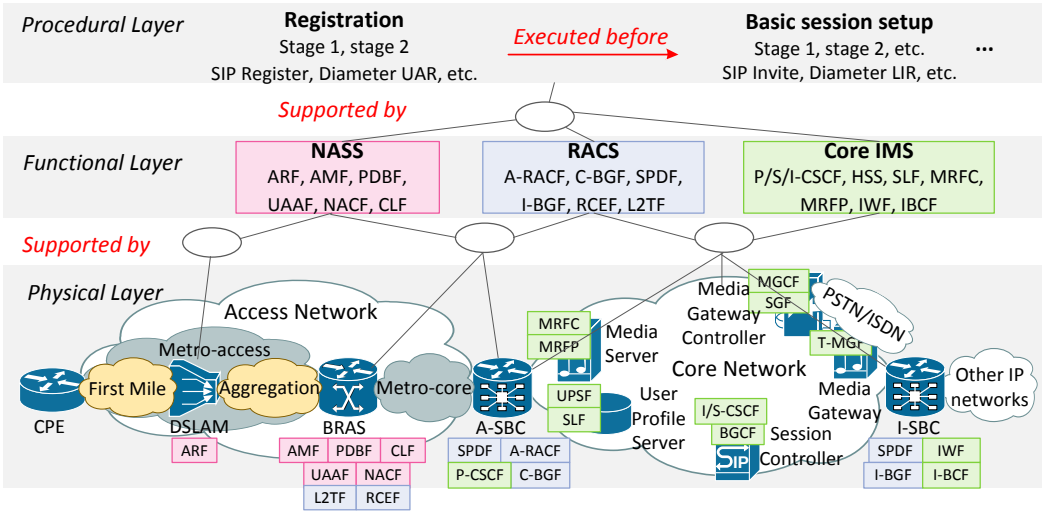


Figure 1.1: Les réseaux IMS sont organisés de façon hiérarchique.

La Figure 1.1 illustre l'architecture générique d'un réseau IMS, qui peut être organisé en trois couches, chacune regroupant les ressources d'une nature spécifique. Le terme « ressource » couvre à la fois les équipements physiques, les logiciels qui s'exécutent à l'intérieur, mais aussi les procédures permettant d'accéder aux services.

La couche physique comprend un *access network* et un *core network*. L'access network est constitué des segments *metro-access* et *metro-core*. Le segment metro-access se décompose en segments plus petits: *first mile* et *aggregation*. Les segments aggregation et metro-core sont connectés via un ou plusieurs Broadband Remote Access Server (BRAS). Le segment metro-core est connecté au core network, contenant la plateforme de service IMS, via des routeurs comme le Session Border Controller (SBC). L'architecture de la couche physique est essentiellement hiérarchique: cette couche est constituée d'équipements spécifiques connectant différents segments réseau, ceux-ci étant décomposables en d'autres équipements physiques connectant des segments réseau plus petits.

La couche fonctionnelle fait référence à l'architecture fonctionnelle IMS et est constituée de trois sous-systèmes. Le sous-système d'attachement au réseau, Network Attachment Subsystem (NASS) (TISPAN, 2010a) fournit l'enregistrement au niveau accès et l'initialisation du terminal utilisateur (User Equipment, UE) pour accéder aux services multimédia. Le sous-système de réservation des ressources et de contrôle

d'admission, Resource and Admission Control Subsystem (RACS) (TISPAN, 2006) est en charge des fonctions de contrôle, des réservations de ressources et du contrôle d'admission. Enfin, le sous-système core IMS (TISPAN, 2007) est en charge de fournir les services multimédia destinés aux terminaux utilisateur. Les sous-systèmes sont des objets hiérarchiques, comprenant plusieurs fonctions listées à l'intérieur de chaque bloc dans l'architecture fonctionnelle de la Figure 1.1. Ces fonctions communiquent entre elles via des interfaces (également appelées des points de référence). Diverse solutions existent pour implémenter ces fonctions dans les équipements/nœuds physiques et les auteurs dans (Darvishan et al., 2009) comparent différentes implémentations.

La Figure 1.1 reflète un exemple d'implémentation avec un réseau d'accès xDSL. Cette implémentation est modélisée par une relation « is-supported-by » entre une fonction et un équipement physique. C'est un premier exemple de dépendances entre ressources puisque la panne d'un équipement a en général un impact sur les fonctions qu'il héberge. La couche procédurale décrit les procédures impliquées dans les opérations permettant d'accéder aux services IMS. Ces procédures, décrites dans les normes, ont une structure multi-résolution: elles se décomposent en phases, qui se décomposent à leur tour en séquences (ou ordre partiel) de requêtes/réponses entre les éléments fonctionnels du NASS, du RACS et du core IMS (voir Figure 1.3 par exemple). Chaque requête/réponse se traduit par des échanges au-dessus d'interfaces qui obéissent à des protocoles tels que DHCP, SIP ou Diameter. L'exécution d'une procédure positionne des variables d'état, certaines d'entre elles sont observables ou peuvent être testées. Par exemple, l'acquisition d'une adresse IP correcte prouve que l'attachement de l'UE au réseau (via le NASS) s'est bien déroulé (voir Figure 1.3). Les procédures, les phases, et leur composants dépendent les uns des autres, dans le sens où ils sont (partiellement) ordonnés en temps, ce que nous notons par la relation « is-preceded-by ». Par ailleurs, ils sont aussi supportés par (relation « is-supported-by ») les fonctions et les interfaces de la couche fonctionnelle qui exécutent ces procédures. Cette information est facilement accessible à partir des normes et révèle une autre forme de dépendances entre ressources.

L'auto-modélisation: Modèle générique et Instance réseau

Le modèle en trois couches décrit ci-dessus est *générique* puisqu'il définit les différents *types* de ressources réseau et la façon dont celles-ci interagissent et dépendent les unes des autres. Cette description provient principalement des informations contenues dans les normes et des pratiques courantes des opérateurs télécoms. Néanmoins, certaines informations doivent être définies par un expert. Ce travail est facilité par la taille relativement petite de ce modèle générique, par l'existence d'une grammaire définissant la façon dont les objets peuvent être liés les uns aux autres, et par la nature hiérarchique du modèle. Cette hiérarchie permet de modéliser chaque couche à différents niveaux de détail (d'abstraction), et ainsi de sélectionner la granularité la plus fine à laquelle on souhaite gérer le réseau. Le modèle générique définit les composants de base d'un réseau, du point de vue de l'activité de gestion. Cependant, le réseau que l'on doit gérer contient plusieurs *instances* de ces composants. Par analogie avec la programmation

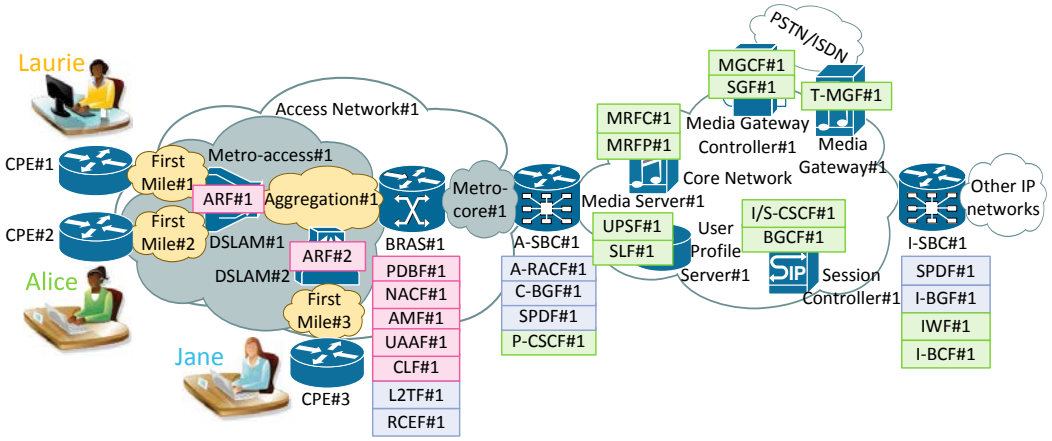


Figure 1.2: Une instance de réseau IMS: la couche physique et les fonctions de la couche fonctionnelle hébergées sont illustrées.

orientée objet, le modèle générique est un diagramme de classes tandis que l'on doit gérer un diagramme d'objets. Le réseau peut être représenté comme une structure à grande échelle où les instances des *patterns* décrits dans le modèle générique partagent certaines ressources et se chevauchent partiellement. Cela est illustré sur la Figure 1.2 où plusieurs utilisateurs d'un réseau d'accès ont des ressources privées (CPE et First mile), mais peuvent partager ou non un DSLAM, un segment aggregation, etc.

Obtenir le modèle de l'instance réseau à gérer signifie créer autant d'instances de ressources réseau (équipements, fonctions, etc) que celles-ci sont décrites dans le modèle générique. Cela signifie aussi structurer ou connecter ces instances conformément aux *patterns* permis par le modèle générique. Une telle construction garantit à la fois l'adéquation du modèle obtenu avec les normes, et permet d'obtenir un modèle adapté à un réseau spécifique, ce qui est d'une importance capitale pour capturer les architectures évolutives. Nous appelons ce processus l'*auto-modélisation*. Cette tâche peut en effet être automatisée à condition que l'architecture de gestion fournisse les outils permettant d'explorer le réseau et de révéler son architecture. Un tel « service » ou une telle « propriété de réflexivité » du réseau, est l'une des fonctions essentielles que l'on doit attendre d'une architecture de gestion autonomique, de même que la capacité d'interroger les éléments gérés dans le but de vérifier leur variables d'état ou de réaliser des tests.

1.4 L'auto-modélisation: support de la localisation de pannes

Méthodologie

La section précédente a illustré un trait typique de la conception des réseaux: des ressources de bas niveau sont assemblées pour construire des ressources de niveau supérieur. Le terme ressource fait référence à des composants de la couche physique,

fonctionnelle ou procédurale. Les relations de dépendances illustrées dans la section précédente nous ont naturellement orienté vers une formalisation en termes de réseaux Bayésiens. Ceux-ci conviennent particulièrement pour représenter à la fois des contraintes et des dépendances statistiques. Par ailleurs, ils encodent les relations d'indépendance conditionnelle sur lesquelles reposent les algorithmes d'inférence, un sujet traité par plusieurs travaux de recherche. Dans ce contexte, l'inférence consiste à inférer la valeur de certaines variables d'état étant donné la valeur observée chez d'autres variables.

Cependant, ce contexte a besoin d'adaptations, et ce, dans plusieurs directions. Tout d'abord, le réseau géré évolue avec le temps (les utilisateurs s'attachent au réseau, s'enregistrent, se déconnectent, de nouveaux équipements sont ajoutés, etc.) et peut ne pas être connu entièrement et dans tous ses détails. Donc, le réseau Bayésien utilisé pour l'inférence devrait être construit sur demande, capturant l'état du réseau, pour répondre à une requête de diagnostic donnée.

Par ailleurs, toutes les ressources réseau ne sont pas impliquées dans le dysfonctionnement d'un service. Donc, seulement une partie du réseau devrait être prise en compte.

Ensuite, cette construction du modèle de réseau Bayésien devrait être couplée avec le moteur d'inférence. Les utilisateurs partagent certaines ressources réseau, par conséquent ils transportent de l'information au sujet de l'état de celles-ci, ce qui peut être utile pour le raisonnement. Donc, l'on doit concevoir une construction dynamique du réseau Bayésien modélisant le réseau, ou, de façon similaire, une exploration dynamique du réseau, pour collecter progressivement des informations et répondre à une requête de diagnostic donnée.

Enfin, voyons ce qu'est une requête de diagnostic. Supposons que l'état d'une certaine ressource est observée comme étant en panne (par exemple la configuration IP est défectueuse pour un terminal utilisateur spécifique), l'on doit découvrir l'origine (la cause primaire) de cette panne. Cette cause primaire se trouve nécessairement dans les ressources de couches inférieures qui sont assemblées pour construire la ressource défectueuse. Ceci peut être vu comme un problème d'inférence d'état étant donné les valeurs observées sur les autres variables d'état.

Par extension, on pourrait imaginer interroger l'état de *toutes* les variables d'un *type* donné, étant donné des observations. Un exemple de requête serait alors la suivante: étant donné la panne d'un segment *first mile*, quelle est la probabilité qu'un UE (non spécifié) soit capable d'effectuer des appels. Ce type de requête d'inférence générique est nouveau dans le formalisme des réseaux Bayésiens, et constitue de toute évidence une approche pour évaluer l'impact des pannes.

La méthodologie que nous proposons pour localiser l'origine d'un dysfonctionnement observé au niveau d'une ressource est la suivante:

1. Retrouver et/ou assembler (à un niveau de granularité donné) le modèle générique (ou réseau Bayésien générique) qui décrit les ressources utilisées par la ressource défectueuse.

2. Localiser l'instance de ce réseau Bayésien générique dans l'instance du réseau IMS

et obtenir ainsi une instance de réseau Bayésien, que nous appelons *pattern*.

3. Au sein du réseau Bayésien actuel (l'instance), alimenter le moteur d'inférence avec les observations disponibles pour localiser la ressource défectueuse.

4. Si les observations récoltées ne sont pas suffisantes, étendre le réseau Bayésien actuel en explorant d'autres patterns (d'autres instances) qui partagent des ressources avec le réseau Bayésien actuel. Dans ce réseau Bayésien étendu, collecter les nouvelles observations disponibles pour améliorer la précision sur localisation de la panne.

5. Répéter l'extension jusqu'à ce que l'origine de la panne soit localisée avec précision.

Exemple

Pour illustrer cette méthodologie sur un cas pratique, supposons que nous souhaitons expliquer pourquoi la configuration IP a échoué pour l'utilisateur Laurie dans l'instance réseau de la Figure 1.2. Comme le montre la Figure 1.3, le service de configuration IP se décompose en deux phases successives, déclenchées par l'UE. Dans la couche fonction-

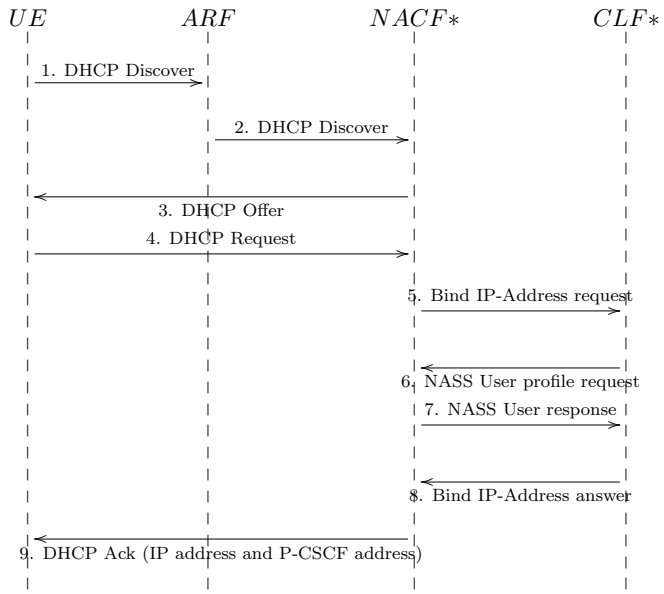


Figure 1.3: Diagramme de séquence de la configuration IP en utilisant le protocole DHCP (couche procédurale). Cette configuration se traduit par un dialogue entre l'UE (à gauche) et les fonctions du NASS (à droite).

nelle, la granularité choisie distingue une ressource individuelle notée « NACF* » qui regroupe les fonctions AMF (Access Management Function), NACF (Network Access Configuration Function), UAAF (User Access Authorization Function), et les interfaces entre elles. De la même façon, le niveau de granularité choisi distingue une ressource

individuelle notée « CLF* » qui regroupe les fonctions CLF (Connectivity session Location and repository Function) et A-RACF (Access-Resource and Admission Control Function), et l'interface entre ces deux fonctions.

En suivant la méthodologie décrite ci-dessus, nous commençons par retrouver le modèle générique qui décrit les ressources utilisées par le service de configuration IP (Figure 1.4). Ce graphe de dépendances entre ressources peut être vu comme un réseau Bayésien, avec des dépendances statistiques si ces statistiques sont disponibles, ou des dépendances logiques sinon. Il traduit, par exemple, le fait que le résultat de la procédure de configuration IP dépende de l'issue du « Stage 1 » qui dépend lui-même de l'état de l'interface entre les fonctions UE et ARF (notée ici « int UE-ARF »). Cet état dépend à son tour du lien physique entre l'UE et l'ARF, c'est-à-dire du segment first-mile. Dans ce réseau Bayésien générique, la variable d'état « IP configuration » est observable puisqu'on peut facilement tester si l'UE a obtenu une adresse IP correcte.

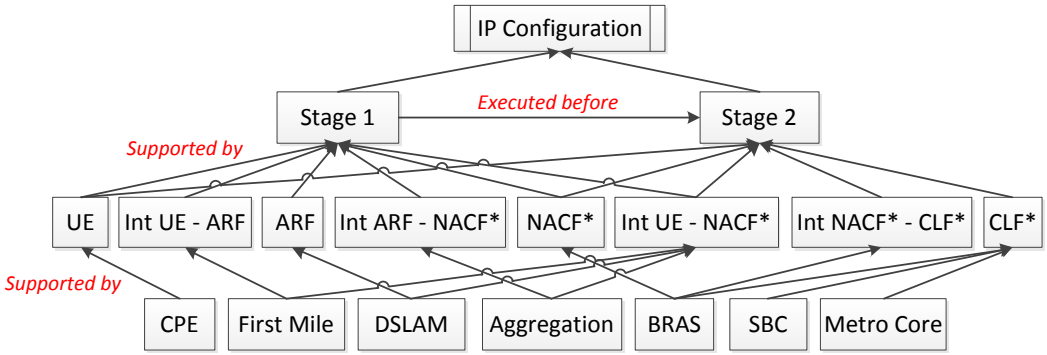


Figure 1.4: Réseau Bayésien générique pour le service de configuration IP d'un terminal utilisateur.

Dans un deuxième temps, nous localisons parmi plusieurs instances de ce réseau Bayésien générique, l'instance qui concerne l'utilisateur Laurie. Cette instance de réseau Bayésien est montrée à la Figure 1.5 où les ressources privées de Laurie sont affichées en orange. Dans l'instance du réseau Bayésien relative à Laurie, l'état de la variable observable « IP configuration » est « down » c'est-à-dire défectueux. Chacun

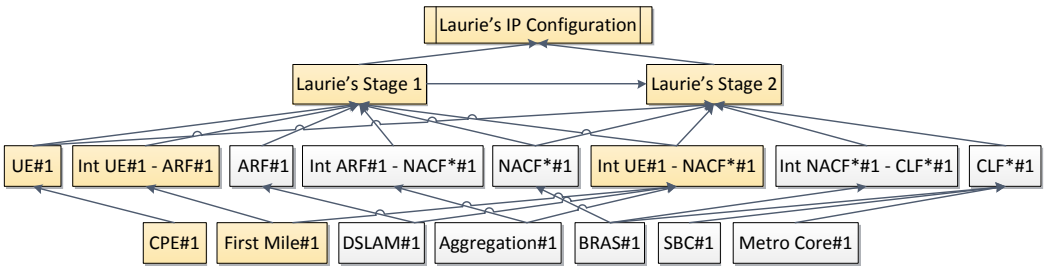


Figure 1.5: L'instance du réseau Bayésien relative à Laurie

des nœuds dans le réseau bayésien de Laurie peut être une explication pour la panne

observée chez Laurie.

La troisième étape consiste à collecter les observations disponibles sur toutes ces ressources, et à exécuter une inférence bayésienne pour essayer de localiser l'origine de la panne. Notons que toutes les ressources ne peuvent pas fournir d'observables sur leur état et, que certaines observations peuvent ne pas être totalement discriminantes.

Dans un quatrième temps, on peut décider de vérifier les explications possibles découvertes jusque-là en interrogeant d'autres utilisateurs qui partagent certaines ressources avec Laurie. Par exemple, nous avons choisi de vérifier l'état de la configuration IP de Jane. Par conséquent, nous augmentons l'étendue du réseau Bayésien actuel (l'instance de réseau Bayésien relative à Laurie) en ajoutant l'instance de Jane. La Figure 1.6 montre l'étendue du réseau Bayésien, qui contient à présent plus de variables observables. Supposons que l'état de la configuration IP de Jane est « up », c'est-à-dire fonctionne correctement. Cette observation implique que toutes les ressources que Laurie partage avec Jane fonctionnent correctement, ce qui aurait été révélé par une inférence Bayésienne sur ce réseau étendu. Par conséquent, l'ensemble des causes (ressources défectueuses) possibles dans l'instance de Laurie est réduit au sous ensemble de ressources qui ne sont pas partagées avec Jane.

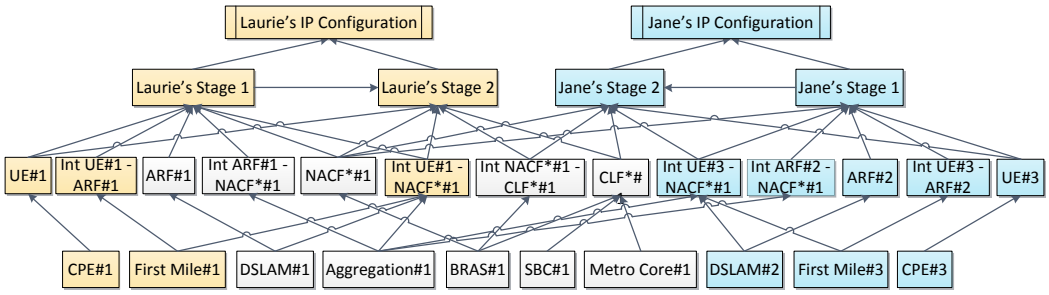


Figure 1.6: L'instance de Jane est ajoutée; ses ressources privées sont représentées en bleu.

Ce processus peut être itéré (point 5): on peut interroger un autre utilisateur, à condition qu'il ou elle partage des ressources soit avec Laurie, soit avec Jane. Mais, le choix de l'utilisateur à interroger devrait dépendre de la valeur informative des nouvelles observations pour localiser la ressource défectueuse qui a causé le problème de Laurie. Si nous choisissons d'interroger Alice, qui a plus de ressources en commun avec Laurie qu'avec Jane, le réseau Bayésien actuel doit encore être étendu pour incorporer l'instance, du modèle générique de configuration IP, relative à Alice. La Figure 1.7 montre le réseau Bayésien étendu (les ressources privées de Jane n'ont pas été montrées). Supposons que la configuration IP d'Alice est « up », c'est-à-dire fonctionne correctement. Cette observation implique que toutes les ressources que Laurie partage avec Alice fonctionnent bien. Un fois de plus, dans l'instance de Laurie, l'ensemble des ressources défectueuses possibles est réduit au sous-ensemble des ressources qui ne sont pas partagées avec Alice ou Jane. Par contre, supposons que la configuration IP pour Alice est « down ». Cette observation augmente notre certitude dans l'état « down » pour les ressources que Laurie partage avec Alice et non avec

Jane. L'explication de la panne observée chez Laurie se trouve probablement parmi ces ressources partagées.

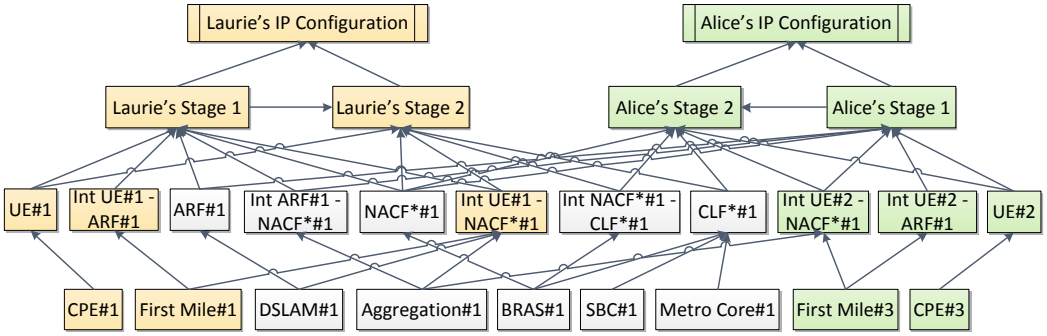


Figure 1.7: L'instance du réseau Bayésien relative à Alice est ajoutée; ses ressources privées sont représentées en vert. Les ressources privées de Jane ne sont pas montrées.

1.5 Raisonnement avec les réseaux Bayésiens génériques

Les sections précédentes ont montré comment les ressources réseau dépendent les unes des autres, formant un grand graphe de dépendances. Ce graphe est modélisé comme un réseau Bayésien où plusieurs parties sont isomorphes, c'est-à-dire que ce graphe est obtenu en connectant des fragments qui sont des copies de modèles génériques. Dans cette section, nous formalisons la construction du réseau Bayésien et expliquons comment réaliser l'inférence au-dessus de ce réseau, en explorant seulement la portion du réseau Bayésien qui est la plus informative vis-à-vis d'une requête de diagnostic donné.

Formalisation

Définition 1.1 *Un réseau Bayésien $X = (V, E, \mathbb{P}_X)$ est formé d'un graphe orienté acyclique $G = (V, E)$, avec V l'ensemble des sommets, $E \subseteq V \times V$ l'ensemble des arcs, et une collection de variables aléatoires $(X_v)_{v \in V}$ indexées par V avec une distribution de probabilité $\mathbb{P}_X$. En notant ${}^\bullet v = \{u \in V : (u, v) \in E\}$ les parents du nœud $v \in V$ dans le graphe (V, E), la distribution de X s'exprime comme suit: $\mathbb{P}_X = \otimes_{v \in V} \mathbb{P}_{X_v | X_{{}^\bullet v}}$, où $X_U = (X_u, u \in U)$ représente un vecteur de variable aléatoires, $U \subseteq V$.*

Comme c'est le cas habituellement, un morphisme $\phi : G_1 \to G_2$ entre graphes orientés acycliques $G_i = (V_i, E_i)$ est une fonction partielle de V_1 vers V_2 préservant les arcs: $\forall u, v \in Dom(\phi) = V_1' \subseteq V_1$, $(u, v) \in E_1 \Leftrightarrow (\phi(u), \phi(v)) \in E_2$. ϕ est appelée insertion of G_2 dans G_1 si et seulement si ϕ restreint à son domaine est bijective, *i.e.* $\phi_{|V_1'}$ est un isomorphisme entre $G_{1|V_1'} = (V_1', E_1 \cap V_1' \times V_1')$ et G_2.

Définition 1.2 *Un réseau Bayésien générique est un ensemble fini de réseaux Bayésiens ordinaires $(W^k)_{1 \leq k \leq K}$, où chaque réseau Bayésien $W^k = (V_k, E_k, \mathbb{P}_{W^k})$ est aussi ap-*

pelé un pattern. Une instance $X = (V, E, \mathbb{P}_X, (\phi_{k,i})_{k \leq K, i \in I_k})$ *de ce réseau Bayésien générique est un réseau Bayésien standard* $(V, E, \mathbb{P}_X)$ *où*

1. *chaque* $\phi_{k,i}$ *est une insertion du pattern* (V_k, E_k) *dans* (V, E)*, et le graphe orienté acyclique* (V, E) *est couvert par de telles instances de pattern:* $V = \cup_{k,i} Dom(\phi_{k,i})$*,* $\forall (u, v) \in E$*,* $\exists \phi_{k,i} : u, v \in Dom(\phi_{k,i})$*.*

2. *la probabilité* $\mathbb{P}_X$ *est héritée des patterns* W_k *à travers les insertion-morphismes* $(\phi_{k,i})_{k \leq K, i \in I_k}$*:* $\forall v \in V$*, on a*

   a) *si* ${}^\bullet v = \emptyset$*, alors* $\forall k, i : \phi_{k,i}(v) = u \;\Rightarrow\; \mathbb{P}_{X_v} \equiv \mathbb{P}_{W_u^k}$

   b) *si* ${}^\bullet v \neq \emptyset$*, alors* $\forall k, i : \phi_{k,i}(v) = u, {}^\bullet v \cap Dom(\phi_{k,i}) \neq \emptyset \;\Rightarrow\; {}^\bullet v \subseteq Dom(\phi_{k,i})$*,* $\mathbb{P}_{X_v | X_{{}^\bullet v}} \equiv \mathbb{P}_{W_u^k | W_{{}^\bullet u}^k}$

En d'autres termes, pour construire X on prend plusieurs copies de différents patterns W^k, et on les agrège en partageant certaines variables aléatoires. Pour assurer la cohérence de cette construction, chaque variable X_v appartenant à différentes instances de pattern doit être définie par la même probabilité conditionnelle. La Figure 1.8 donne un exemple d'un réseau Bayésien générique avec un seul pattern de cinq variables, et une instance de ce réseau Bayésien générique avec six copies du pattern se chevauchant de différentes façons. La localisation de pannes dans une instance de réseau Bayésien

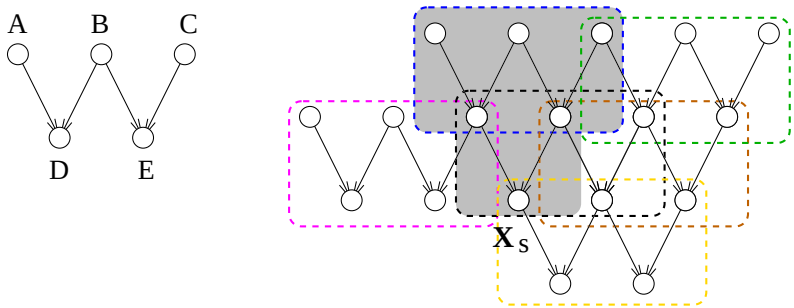


Figure 1.8: Un réseau Bayésien générique (à gauche) avec un pattern unique, et une instance de ce réseau (à droite) avec six instances de ce pattern se chevauchant.

générique peut être traduite en un problème d'inférence classique de la façon suivante. Supposons qu'une variable, par exemple X_s dans la Figure 1.8, est déclarée/observée comme défectueuse. Cette ressource X_s dépend d'autres ressources, à savoir $X_{{}^{\bullet\bullet}s}$ où ${}^{\bullet\bullet}s$ désigne tous les ancêtres du nœud s. Soit la panne de X_s est spontanée, soit, elle résulte d'un pattern de propagation de pannes dans $X_{{}^{\bullet\bullet}s}$. L'objectif est donc d'estimer la valeur de X_Z, avec $Z = {}^{\bullet\bullet}s \cup \{s\}$ (en gris dans la Figure 1.8). A défaut on pourrait construire une localisateur de cause primaire L, comme une fonction de X_Z qui contraint la présence d'une unique cause primaire au sein de X_Z. L'objectif est alors d'estimer la valeur de L.

Il existe des variables observables au sein de (ou directement attachées à) de X_Z, en particulier l'observation indiquant la valeur défectueuse de X_s. Désignons par

$Y_0 = y_0$ ce vecteur observé. La loi conditionnelle $\mathbb{P}_{X_Z|Y_0=y_0}$ donne une première information de ce qui a causé la panne de X_s: par maximum de vraisemblance, on obtient une explication (c'est-à-dire un schéma de propagation de pannes), la valeur $\hat{X}_Z = \arg\max_x \mathbb{P}_{X_Z|Y_0=y_0}(x)$. On peut évaluer à quel point cette explication est fiable en calculant l'entropie conditionnelle de X_Z étant donné $Y_0 = y_0$:

$$H(X_Z|Y_0 = y_0) = \sum_x -\mathbb{P}_{X_Z|Y_0=y_0}(x) \cdot \log_2 \mathbb{P}_{X_Z|Y_0=y_0}(x)$$

Une grande valeur d'entropie conditionnelle signifie que les observations $Y_0 = y_0$ sont insuffisantes pour choisir parmi plusieurs explications. On peut par conséquent chercher à collecter plus d'observations dans le but d'obtenir plus d'information sur X_Z, c'est-à-dire réduire l'entropie conditionnelle. Plus cette entropie tend vers zéro, plus l'estimation de X_Z est fiable. L'idée est d'explorer une zone plus large de l'instance du réseau Bayésien dans le but de collecter plus d'observations, comme l'illustre l'exemple de la section précédente. A partir de l'ensemble de nœuds explorés $U_0 = Z$, on passe à $U_1 \supseteq U_0$ tel que U_1 soit fermé pour la relation d'ancêtre: ${}^{\bullet}U_1 \subseteq U_1$. Cela définit un nouvel ensemble d'observations $Y_1 = y_1$, à partir duquel $\hat{X}_Z$ peut être estimé à condition que $H(X_Z|Y_0 = y_0, Y_1 = y_1)$ soit suffisamment faible. Sinon, on poursuit avec une autre extension.

S'agissant de la Figure 1.8, on peut étendre la partie explorée de l'instance du réseau Bayésien pour capturer des observations soit dans l'instance de pattern verte, ou dans celle de couleur violet. Désignons par Y_1 et Y_2 ces deux ensembles possibles d'observations. L'ensemble le plus informatif est obtenu en comparant $H(X_Z|Y_0 = y_0, Y_1)$ à $H(X_Z|Y_0 = y_0, Y_2)$: la valeur la plus faible indique l'ensemble le plus informatif en moyenne. Cette valeur est calculée au préalable, sans interroger/tester la valeur effective de l'observable Y_i sélectionnée. Supposons que Y_1 est la plus prometteuse. Cela définit une nouvelle zone U_1 prise en compte dans l'instance du réseau Bayésien, où on peut collecter la valeur observée $Y_1 = y_1$. Après une inférence classique, on obtient la distribution à postériori $\mathbb{P}_{X_Z|Y_0=y_0,Y_1=y_1}$ et par conséquent l'entropie conditionnelle $H(X_Z|Y_0 = y_0, Y_1 = y_1)$. Notons que $H(X_Z|Y_0 = y_0, Y_1 = y_1)$ peut en fait être plus petit ou plus grand que la valeur moyenne $H(X_Z|Y_0 = y_0, Y_1)$ utilisée pour déterminer l'observation la plus prometteuse.

L'introduction de nouvelles mesures se poursuit soit jusqu'à ce que l'entropie conditionnelle devienne suffisamment faible, soit jusqu'à ce qu'il n'y ait plus d'observation disponible.

1.6 Résultats

Pour démontrer la pertinence de la stratégie d'exploration ci-dessus, nous avons réalisé des tests sur une grande instance de réseau Bayésien générique en forme d'arbre. L'unique pattern correspond à celui de la Figure 1.8 où les variables X_D et X_E sont observables. L'instance est illustrée à la Figure 1.9, où chaque sommet représente une instance du pattern (donc 5 variables). Deux instances de patterns connectées se

chevauchent en partageant 1, 2 ou 3 des variables parmi les nœuds $\{A, B, C\}$, ce qui se reflète par l'épaisseur de l'arrête connectant ces patterns. Dans l'instance de pattern X_1, les variables $X_{1,A}, X_{1,B}, X_{1,C}$ sont des variables binaires uniformes indépendantes. Dans tous les autres patterns X_i, $i > 1$, les variables $X_{i,A}, X_{i,B}$ ou $X_{i,C}$ nouvellement crées ont une distribution $(p, 1-p)$ avec $p = 0.9$, dans le but de garantir une corrélation à longue portée entre les instances de pattern présentées à la Figure 1.9. Dans le pattern X_i, les observations Y_i correspondent aux variables $X_{i,D}, X_{i,E}$. D est plus sensible à une panne de B qu'à une panne de A. De même, E réagit plus à C qu'à B. Pour les deux capteurs, le taux de faux positif et le taux de faux négatif est de 5%. L'expérience

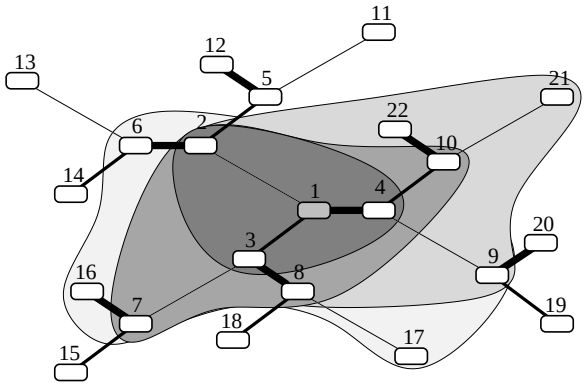


Figure 1.9: Une instance de réseau Bayésien générique avec 22 instances de pattern. L'épaisseur de la ligne reliant deux instances reflète le nombre de variables partagées. Pour estimer le pattern central X_1, les observations sont introduites par zones, en commençant par la zone la plus foncée et en poursuivant vers les zones plus pales.

consiste à tirer un échantillon aléatoire du processus décrit par cette instance de réseau Bayésien générique, et à calculer la distribution conditionnelle $\mathbb{P}_{X_1|Y_U=y_U}$ du pattern central étant donné un ensemble de mesures croissant U. Cet ensemble de mesures part de $U = \{1\}$ jusqu'à $U = \{1, 2, ..., 22\}$. Entre temps, la mesure la plus informative est incorporée à chaque étape. Pour chaque échantillon, l'évolution de $H(X_1|Y_U = y_U)$ a été calculée. L'expérience a été menée un millier de fois. En moyenne, on observe que $H(X_1|Y_U = y_U)$ décroit (Figure 1.10) et converge rapidement. Cependant, pour un échantillon donné, cette courbe peut ne pas décroissante, en effet, même si une observable est très informative en moyenne, la valeur réelle observée peut mener à une révision des hypothèses courantes, par conséquent à une augmentation temporaire de l'incertitude. Autre détail intéressant, nous avons réalisé des statistiques de rang sur la façon dont les mesures sont introduites. En moyenne, le meilleur ordre s'avère être 1, 3, 4, 2, 7, 10, 8, 9, 21, 22, 17, 6, 16, 5, 13, 15, 20, 18, 19, 12, 14, 11, ce qui est reflété par les zones grandissantes autour de X_1 dans la Figure 1.9. Les patterns fortement couplés ont tendance à être favorisés, mais cela n'est pas toujours le cas.

Nous avons également démontré la pertinence de cette approche pour résoudre un scénario typique de panne dans une architecture IMS. Ce scénario ainsi que les résultats correspondants sont décrits au chapitre 8 de cette thèse.

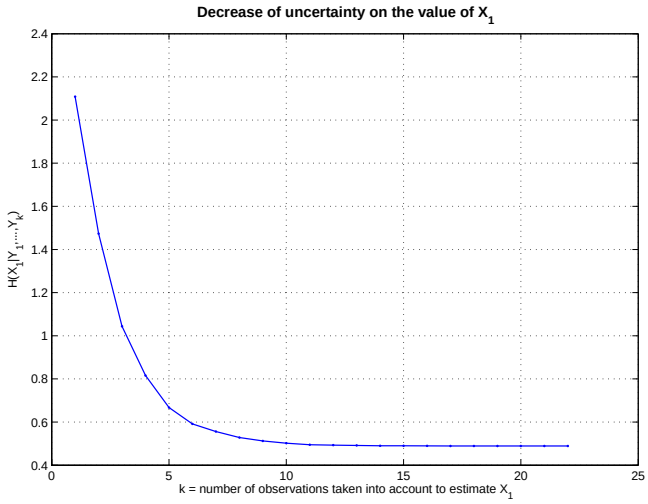


Figure 1.10: Evolution en moyenne de l'entropie conditionnelle du pattern central X_1 au fur et à mesure que le nombre d'observations augmente.

1.7 Conclusion

Les techniques basées modèle sont la clé pour gérer de façon ambitieuse et autonome les pannes survenant dans un réseau. Elles s'adaptent à des instances réseau, offrent un large éventail de techniques de raisonnement, suggèrent des méthodes pour analyser l'impact des pannes et peuvent aller plus loin en suggérant des mesures de réparation. Leur talon d'Achille réside dans l'obtention d'un modèle précis du réseau géré. Nous avons proposé un nouveau concept: l'auto-modélisation qui réduit la difficulté de la construction du modèle à la définition d'un nombre limité de patterns génériques du modèle, en exploitant la connaissance disponible dans les normes. Le modèle réel correspondant à une instance réseau donnée est ensuite construit en connectant autant de copies (c'est-à-dire d'instances) de ces patterns génériques que nécessaire. Tout cela, dans le but de reproduire la structure de dépendances entre les ressources déployées dans cette instance de réseau.

Nous avons proposé un formalisme basé sur la notion de réseau Bayésien générique: un réseau Bayésien composé d'un certain nombre de copies de patterns. Même si l'instance du modèle est large, on a seulement besoin d'explorer la partie de cette instance nécessaire pour expliquer/diagnostiquer un disfonctionnement observé. En effet les variables trop éloignées transportent peu d'information au sujet d'un dysfonctionnement observé. Cependant, ce formalisme a besoin d'être complété pour capturer la construction hiérarchique intrinsèque aux réseaux, c'est-à-dire le fait qu'un segment réseau se décompose généralement en une structure de segments réseaux plus petits, et cela, de façon récursive. De tels réseaux Bayésiens génériques et multi-résolution n'existent pas encore, mais semblent être cruciaux pour la gestion des réseaux. Une autre direction de recherche serait le traitement simultané et/ou successif de plusieurs

requêtes de diagnostic. Une autre direction encore consisterait à réaliser ce raisonnement de façon distribuée pour capturer le fait que les segments réseau sont généralement gérés par différentes unités opérationnelles. Enfin, le calcul d'impact des pannes, peut être réalisé en évaluant l'impact d'une panne donnée sur toutes les variables d'un certain *type* dans un pattern générique donné du réseau Bayésien.

Liste des publications

- (Hounkonnou and Fabre, 2012) Carole Hounkonnou and Eric Fabre. Empowering Self-diagnosis with Self-modeling. In *8th international conference on Network and Service Management (CNSM)*, CNSM 2012, pages 364–370, 22–26 October 2012 in Las Vegas.
- (Hounkonnou and Fabre, 2013) Carole Hounkonnou and Eric Fabre. Enhanced OSPF graceful restart. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, IM 2013, 27–31 May 2013 in Belgium.
- (Hounkonnou et al., 2012) Carole Hounkonnou, Samir Ghamri-Doudane, and Eric Fabre. Détection et correction de boucles de routage dans un réseau de routeurs utilisant un protocole de routage de type SPF. *Europeean Patent*, 2012.

Contents

1 Résumé en français **3**
1.1 Contexte . . . 3
1.2 Introduction . . . 4
1.3 Structure des réseaux IMS . . . 6
1.4 L'auto-modélisation: support de la localisation de pannes . . . 8
1.5 Raisonnement avec les réseaux Bayésiens génériques . . . 13
1.6 Résultats . . . 15
1.7 Conclusion . . . 17

2 Introduction **21**
2.1 Scope . . . 21
2.2 Objective . . . 24
2.3 Thesis structure . . . 25

3 Enhanced OSPF Graceful Restart **27**
3.1 Normal and Graceful restarts in OSPF . . . 28
3.2 Properties of routing graphs . . . 30
3.3 Prediction of routing loops . . . 32
3.4 Correction of a routing loop . . . 33
3.4.1 Severity Degree of Routing Loops . . . 33
3.4.2 Correction of a Routing Loop . . . 35
3.4.3 Scheduling of Backup Routings . . . 36
3.5 Correction of multiple routing loops . . . 38
3.6 Evaluation of the enhanced graceful restart . . . 39
3.7 Discussion . . . 41

4 State of the art **43**
4.1 Rule-based Expert systems . . . 46
4.2 Model-based Expert systems . . . 51
4.3 Case-based Reasoning systems . . . 54
4.4 Knowledge discovery and data mining . . . 56
4.5 Fault-symptom graphs and the codebook approach . . . 64
4.6 Dependency/causality graphs . . . 68
4.7 Chronicles: patterns of timed events . . . 73
4.8 Bayesian Networks . . . 78
4.9 Summary . . . 87

5 Structure of IMS networks **89**
5.1 Physical layer of the IMS Architecture 89
5.2 Functional layer of the IMS architecture 90
5.2.1 Core IMS subsystem 91
5.2.2 RACS subsystem 95
5.2.3 NASS subsystem 97
5.3 Procedural layer of the IMS Architecture 101
5.3.1 NASS procedures 102
5.3.2 RACS procedures 104
5.3.3 Core IMS procedures 105
5.4 Contributions: Relevant structural properties 111

6 Network model describing resource dependencies **113**
6.1 Different types of resources and their relationships 113
6.1.1 Physical resources and their relationships 114
6.1.2 Functional resources and their relationships 115
6.1.3 Procedural resources and their dependencies 117
6.1.4 Inter-layer relationships 118
6.2 Contributions: Models of network dependencies 120
6.2.1 Network model class hierarchy 120
6.2.2 Causal graph of network dependencies 121
6.2.3 Generic model vs network instance: self-modeling 123

7 Self-Modeling as Support for Fault Localization **125**
7.1 Methodology 126
7.2 Generic Bayesian Networks 131
7.3 Fault localization in a GBN instance 132

8 Experimental results **135**
8.1 Entropy, relative entropy and mutual information 135
8.1.1 Entropy and conditional entropy 135
8.1.2 Relative entropy and mutual information 136
8.1.3 Gain function for fault localization 138
8.2 Implementation and evaluation 139
8.2.1 Entropy reduction as more measurements are collected 139
8.2.2 Fault localization in an IMS network 141

9 Conclusion **151**

Bibliography **164**

Table des figures **165**

Glossary **169**

Chapter 2

Introduction

Contents

2.1 Scope **21**
2.2 Objective **24**
2.3 Thesis structure **25**

2.1 Scope

Telecom operators are going through a technological and business revolution. In addition to existing services such as telephony or leased line services, spread of the Internet, the Internet Protocol (IP) phone, and new communications services like IPTV are making great progress with the development of digital subscriber lines (DSL) and high-speed communications technologies like Fiber-to-the-home (FTTH). Furthermore, with the deployment of Next Generation Networks (NGNs), development of still newer services is anticipated. Communications networks developed over the last two decades have profoundly changed the way we carry out our everyday lives—how we exchange information, engage in commerce, form relationships, entertain ourselves, protect ourselves, create art, learn, and work. The emerging world is pervasive and strives towards integrating people, technology, environment and knowledge. This emerging vision sets the users at the core of the networks. From passive end-points, they became permanent active components of layered and meshed networks and sources of information transferred or accessed worldwide.

Besides these technology and usage revolutions, a change in the rules of the game and in regulations have also led to a mutation in the value chain. Competition, online service and content providers, the apparent free access and use of services, and advertising dispatched over the networks, etc., have created new businesses and business models. This context has incited operators (but also suppliers and others actors) to explore new territories at the boundary of their core business in order to follow the value, the end customer.

At the technical level, historical bottlenecks disappear. Broadband fixed and mobile technologies are now implemented on subscriber connection and are today providing exceptional opportunities for Telecom operators to transform their business and their

infrastructures. These services require closer network and IT, bringing together fixed and mobile infrastructures in order to get new innovative services and cost savings through common service enablers. To reach this goal, network operators have studied and set-up multimedia broadband infrastructures based on a completely new framework. IP has become the universal and common transport protocol for any type of digitalized information. New architecture principles, like the Next Generation Network (NGN) principle of separation of transport and control functions and the IP Multimedia Subsytem (IMS) principle of common control for mobile and fixed services, are enabling control and transport of data flows of any nature and origin. This includes the more stringent ones, i.e., those coming from conversational or real-time TV services.

One can notice new access characteristics: the increasing symmetry of user flows on fixed services, from xDSL over copper to optics, and on mobile services, from Universal Mobile Telecommunications Services (UMTS) to 4th Generation (4G) access as well as widespread implementation of always-on connected user equipment. This technical revolution provides a great opportunity for Telecom operators to share network infrastructures between fixed, mobile, Internet, and content services. It provides the opportunity for separation from legacy networks (PSTN, X25, PDH), thus contributing to medium term cost savings and complexity reduction, even though mass migrations from legacy to new technologies may be costly, painful, and risky. Triple play is voice, Internet, and TV services access. Quadruple play adds mobile services. Tomorrow there will be multiple play services. These are made possible through a single generic broadband access. This is *the* challenge.

Besides proposing higher access throughputs at home, on the move, and at the office, Telecom operators also have a fundamental imperative: to bring a continuous flow of innovation into their networks, services, and IS. This will lead, for instance, to enhancements in content offers (HDTV, 3DTV, mobile TV, etc.) and the daily operation of services, such as health and security. It will support the development of user generated content and social networks to insure better experiences on existing services (VoIP, TV, VoD, etc.) and provide, for a given service, continuity and fluidity abilities on different devices (multi-screen strategy) and access (fixed and mobile). This profusion of technologies and usage has resulted in a tremendous amount of complexity. The need to simplify has become more than evident. This is the reason why convergence, mutualization, architectural efforts and so on, are essential tools to obtain simplicity for service usage as well as service and network operation.

In summary, Telecom operators have a number of challenges to face.

The commercial challenge is that historical business models, based mainly on voice transport, are no longer sustainable. In a world of abundance, protecting a viable business model by driving a broadband-everywhere strategy, while taking advantage of assets and traditional strength, is an essential issue. This includes the use of capabilities such as billing (useful for billing third-party services), business intelligence (profiling, localization, and so on) based on knowledge of their customers, Quality-of-Service (QoS), and customer experience. Historical know-how and lessons are important, pulled from dozens of years of real-time applications delivered to millions of customers.

The technical challenge is to select the best-of-breed of new technologies whose

arrival rate has never been so rapid. The technical challenge is to maintain agility and secure robustness and scalability for new innovative services in a complete IP-based world of transformation. Agility means the ability to evolve service platforms and IT to support faster service rollouts. To secure robustness and scalability means the ability for network and IT architecture and design and implementation to face the growth of traffic and number of customers generated by new services. And last, but not least, to improve customer experience. Triple play/quadruple play is currently under deployment in conjunction with a "broadband everywhere" strategy in fixed and mobile domains (FTTx, HSPA, LTE, WIMAX, etc.). This is going to have deep consequences all along the technical/network chain, from the customer premises (home network), network access, backhaul and aggregation, transport backbone, service and network control, service platform, and finally to IT.

The technical challenge cannot be successfully achieved if network and IT operations challenges, e.g., new operations models and processes, are not addressed and achieved. The key differentiator will be the ability to ensure, day-after-day, the QoS and competitive cost expected by customers. This will have the ability to hide (from customers) the overall complexity. A number of quality problems with triple and quadruple play exist, such as dropped VoIP calls, bad audio or video quality, long IPTV channel zapping delay, and others. In the end, what matters is the quality of experience, the quality as perceived by the customer. This challenge should be pursued while keeping operating expenses (OPEX) under control. This is critical in triple play operation and is valid for service provision, network operation, after sale processes, etc.

The existing network and IT architecture, methods of operation, delivery process (commercial and technical aspects), and operational structure need to be adapted to better fit with the characteristics of new services and business challenges. Unfortunately, network and system management solutions are no more capable to deal with the increasing complexity; they still rely on very expensive and rare human experts to solve problems, which themselves are beyond the capacities of the experts. Many problems also arise from these experts' intervention, such as misconfigurations (wrong configuration, tuning). These misconfigurations are among the most complex problems to solve; they are very difficult both to understand and locate and therefore to fix. Operators now understand that it is vital for them to master this increased, uncontrollable operational cost (OPEX) (including the deployment cost) by deploying breaking approaches.

The only response to this unsustainable situation is innovation in the way networks are managed and controlled. It is necessary to develop new networks that are able to automatically adapt their configurations to the increases and changing requirements of end users and service providers. Soon, we'll see drastic developments in the end users' services with the introduction of high-speed access networks that are either fixed with the deployment of FTTH or wireless with LTE and WiMAX technologies. Future networks need to be more flexible, capable of reorganizing in an autonomic way when new types of equipment or services are introduced, reducing the need for human intervention and consequently the associated costs. Future networks should be able to improve their performances when needed to respond to unusual changes in the traffic

pattern. The innovation should help to design new types of equipments, protocols, and network architectures and even services that are able to be self-managed, to reduce the operational burden on the operators by themselves making decisions in terms of configuration, optimization, and the like.

If networks and services are able to exhibit some level of autonomy that will allow them to themselves solve their problems in any context, then the operator will be able to reduce the need for intervention by human experts and therefore reduce their operational costs (OPEX). It is time that significant progress be made in how to manage and control these complex infrastructures at the early stage of their design. Many initiatives have been launched to push toward innovations in this area. These initiatives have different names, but all converge to the emergence of a new generation of intelligent equipment, networks, and services that are able to exhibit self-* properties. These initiatives are variously named—for example, Autonomic Communication (AC), Autonomic Networks (AN), Automatic Network Management (ANM), Self-Managed Networks (SFN), Situated Networks (SN). Differences in the focus of the various approaches can explain roughly the differences in the terminology, but all of them have one thing in common: they all seek to introduce self-adaptive capabilities in the network, avoiding human interventions as much as possible.

2.2 Objective

We address the challenge of contributing to the realization of some self-management properties in telecommunication networks. Our strategy to achieve this goal is to use a dependency model describing the structure of the network, the relationships between network components/functions, as well as their typical behaviours.

As a first step, we have looked at the maintenance of Modern Open Shortest Path First (OSPF) routers in IP transport networks. These routers can preserve their packet forwarding activity while they reboot. This enables maintenance operations in the control plane with minimum impact on the data plane, such as the Graceful Restart (GR) procedure. This of course assumes the stability of the network topology, since a rebooting router is unable to adapt its forwarding table and may cause routing loops. The Graceful Restart standard thus recommends to revert to a normal OSPF restart as soon as a topological change is advertised. We propose to be less conservative and to take full advantage of the separation between the control and forwarding functions. This is achieved by new specific functionalities:

- the prediction of routing loops caused by a restarting router;
- the determination of the minimal number of temporary backup forwarding actions that should be applied to prevent these loops, without reverting back to a normal OSPF restart;
- the design of action plans to set and remove these temporary backups in order to avoid micro-loops when the restarting router goes back to a normal functioning.

Besides being useful for planning maintenance operations, a model of resource dependencies can be used as a fault propagation model. Based on this observation, in a second phase, we have proposed an approach to automatise the management of faults, covering the different segments of a network, and the end-to-end services deployed over them. This is a model-based approach addressing the two weaknesses of model-based diagnosis namely deriving an accurate model and dealing with huge models. We propose:

- a solution called self-modeling that formulates off-line generic patterns of the model, and identifies on-line the instances of these patterns that are deployed in the managed network;
- an active (self-)diagnosis engine, based on a Bayesian network formalism, that consists in reasoning on a progressively growing fragment of the network model, relying on the self-modeling ability: more observations are collected and new tests are performed until the faults are localized with sufficient confidence.

This active diagnosis approach is experimented to perform cross-layer and cross-segment alarm management on an IMS network.

2.3 Thesis structure

The rest of the thesis is organized as follows.

- **Chapter 3** presents the enhancements we propose to the standardized Graceful Restart procedure. First, it illustrates the normal and graceful restarts of OSPF and explains how routing loops can occur during a graceful restart. Then, the notions of source and destination graphs are introduced. These graphs are central for the detection of routing loops. Next, the severity of such routing loops is characterized, using coloring properties of destination graphs. It is then explained how to correct such loops by temporary reroutings. Finally, we evaluate, on a typical network topology, the proposed enhanced OSPF GR.
- **Chapter 4** reviews some of the numerous contributions to the topic of fault and alarm management. It does not aim to be exhaustive, but rather tries to sample the domain in order to give an overview of the techniques that were proposed and experimented, before discussing their advantages, drawbacks and positioning the ambition of this thesis.
- **Chapter 5** starts from standard descriptions of IMS networks, and identifies the network resources involved in such networks, their structuring and above all their dependencies, which will be the knowledge used to diagnose malfunctions. The main idea is that the failure of a resource is either spontaneous, or results from the failure of a second resource that is necessary to the first one. We propose to represent an IMS network by a dependency model of three multi-resolution layers.

- **Chapter 6** uses object oriented paradigm to represent IMS network resources and their relationships. The three-layer model described in the previous chapter is *generic*, in the sense that it defines the different *types* of network resources involved, how they depend on each other and how they interact. But the actual network one has to manage contains many *instances* of these elements. This actual network can be represented as a large collection of instances of the patterns described in the generic model, and these instances overlap on some common resources.

- **Chapter 7** demonstrates how the resources involved in a network and its services depend on each other, thus forming a huge dependency graph. This graph can be modeled as a Bayesian network (BN) where many parts are isomorphic, *i.e.* this graph is obtained by connecting tiles that are copies of a limited family of generic patterns. Then, we formalize this construction of a possibly large Bayesian network, and explain how to perform inference over it, by exploring only the portion of the BN that is the most informative to a given diagnosis query. Finally, fault localization in a Generic Bayesian Network (GBN) instance is translated into a standard BN inference.

- **Chapter 8** aims at demonstrating the relevance of the exploration strategy explained in the previous chapter. We review some standard results and definitions related to Entropy, Relative Entropy, Mutual Information and Gain Function for fault localization. These concepts and their properties are useful to analyse the experimental results that will be presented.

The final part concludes this thesis and outlines directions for future work.

List of publications

- (Hounkonnou and Fabre, 2012) Carole Hounkonnou and Eric Fabre. Empowering Self-diagnosis with Self-modeling. In *8th international conference on Network and Service Management (CNSM)*, CNSM 2012, pages 364–370, 22–26 October 2012 in Las Vegas.

- (Hounkonnou and Fabre, 2013) Carole Hounkonnou and Eric Fabre. Enhanced OSPF graceful restart. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, IM 2013, 27–31 May 2013 in Belgium.

- (Hounkonnou et al., 2012) Carole Hounkonnou, Samir Ghamri-Doudane, and Eric Fabre. Détection et correction de boucles de routage dans un réseau de routeurs utilisant un protocole de routage de type SPF. *Europeean Patent*, 2012.

Chapter 3

Enhanced OSPF Graceful Restart

Contents

3.1 Normal and Graceful restarts in OSPF 28
3.2 Properties of routing graphs 30
3.3 Prediction of routing loops 32
3.4 Correction of a routing loop 33
3.4.1 Severity Degree of Routing Loops 33
3.4.2 Correction of a Routing Loop 35
3.4.3 Scheduling of Backup Routings 36
3.5 Correction of multiple routing loops 38
3.6 Evaluation of the enhanced graceful restart 39
3.7 Discussion 41

OSPF (Open Shortest Path First) (Moy, 1998b,a) is a widely used link state routing protocol in the Internet. Modern router architectures separate the data plane, and thus the forwarding function, from the control plane, that runs the routing protocols such as OSPF. This creates a possibility to keep forwarding packets while the control plane is being restarted. This so-called Graceful Restart procedure has been standardized (Moy et al., 2003) and is available in commercial routers (Juniper, 2013; Cisco, 2012). Graceful Restart requires the cooperation of all routers neighboring the restarting one. Their role is to keep up the adjacency with the restarting router as long as the topology remains static. In case of any change in the topology, one must immediately stop the graceful restart and return to the standard OSPF behavior, which thus fully removes the restarting router from the topology. This intends to avoid the possible creation of routing loops resulting in packet losses and unreachable destinations.

Such an abrupt change of behavior can be temporarily harmful to the network. And, strictly speaking, it may not be necessary: not every topological change will result into a routing loop, so the forwarding activity of the restarting router could be maintained. Furthermore, even if routing loops are created, they can be temporarily fixed. We study the possibility of such smoother changes of behavior.

Since the standardization of the graceful restart procedure, few papers have examined its practical consequences. (Ghamri-Doudane and Ciavaglia, 2010) examined how a general reboot of all routers could be organized, taking into account that a helper

node cannot reboot until the node it is helping has completed its own reboot. To the best of our knowledge, however, the issue of preventing routing loops during the graceful restart of OSPF routers has only been tackled by Shaikh et al. in (Shaikh et al., 2002) and more recently in (Shaikh et al., 2006). These contributions detail necessary conditions to the existence of routing loops, in the case of several restarting routers, and propose to remove the restarting routers from the forwarding path as soon as these conditions are detected. We follow a similar approach for the detection, but relies on a necessary and sufficient condition for the existence of routing loops, in the case of a single restarting router. The developments then go further by proposing minimal temporary corrections to such loops, and by correcting simultaneously multiple problematic destinations. In our approach, when a routing loop is detected, only a few nodes are informed and apply a correction, rather than broadcasting a global warning to all nodes and returning to a standard OSPF behavior. As a result, the restarting router is maintained in the topology for all destinations to which it is not dangerous.

The chapter is organized as follows. Section 3.1 illustrates the normal and graceful restarts of OSPF and explains how routing loops can occur during a graceful restart. Section 3.2 introduces the notions of source and destination graphs. These graphs are central for the detection of routing loops (Section 3.3). Section 3.4 characterizes the severity of such routing loops, using coloring properties of destination graphs. It then explains in detail how to correct such loops by temporary reroutings, in the case of a single problematic destination. Section 3.5 extends the problem to several problematic destinations to correct simultaneously. Finally, Section 3.6 evaluates, on a typical network topology, the proposed enhanced OSPF Graceful Restart.

3.1 Normal and Graceful restarts in OSPF

OSPF runs on a simple abstract vision of the network: a weighted and directed graph (Figure 3.1), that we call the *topological graph*.

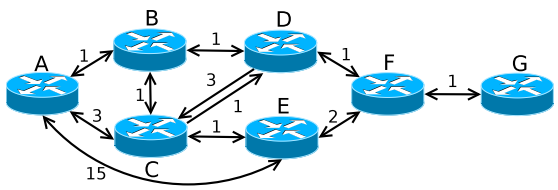


Figure 3.1: An example of OSPF network

At the core of the OSPF routing protocol is a distributed, replicated link state database that describes the collection of routers in the domain, how they are interconnected, and the quality of each link. Each router in the routing domain is responsible for describing its local piece of the routing topology in link-state advertisements, or LSAs. These LSAs are then reliably distributed to all the other routers in the routing domain in a process called flooding. Taken together, the collection of LSAs generated

by all of the routers is called the link-state database. So each node knows the full topological graph at any time. Given the link state database, and assuming this is a reliable description of the network state, each node/router runs Dijkstra's algorithm to derive the shortest paths to all other nodes. The shortest paths originating from (and calculated by) some router R organize as a shortest-paths tree (SPT) rooted at R that we call the *source graph* for router R. Figure 3.2 displays this source graph for node C in the network of Figure 3.1. The SPT defines the routing table, associating a 'next hop' to each destination: for example, at node C, packets to destinations D, F or G will be forwarded to D.

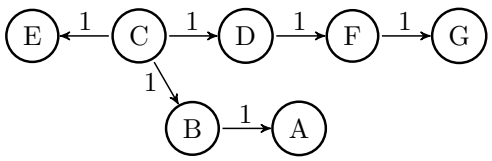


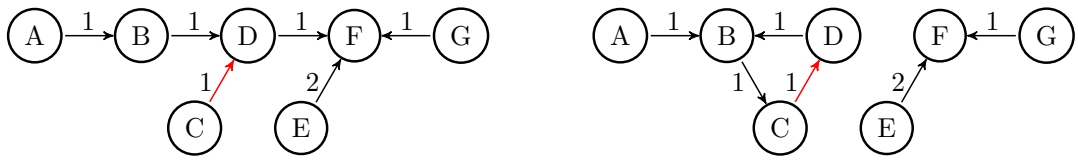
Figure 3.2: Shortest paths from C to all destinations (the source graph of C).

During a normal router restart, the router's neighbors break adjacency with the restarting one, i.e. they generate new LSAs that are flooded throughout the network and cause all routers to update their forwarding tables in order to avoid the rebooting node. A few minutes later, once the restart is completed, the router's neighbors re-establish adjacency with the rebooted one and the whole sequence of LSA floodings and forwarding tables updates is repeated.

With a graceful restart, a router, whose control plane is about to restart and whose forwarding plane functions normally, sends a grace LSA to its neighbors, declaring its intention to perform a graceful restart within a specified grace period. The neighbor nodes (known as helpers) continue to list the restarting router as fully adjacent in their LSAs during the grace period, but only if the network topology remains static. Once the control plane restarts, the restarting router goes through a normal adjacency establishment procedure with all the helpers, at the end of which the restarting router and the helpers regenerate their LSAs.

Any change in the network topology during the grace period would cause the helpers to abort the graceful restart and generate their LSAs showing the breakdown of adjacency with the restarting router. Indeed, the latter is unable to adjust its forwarding table in a timely manner when the network topology changes. Its forwarding table is said to be *frozen*. Since this table may no longer be consistent with the new network topology, routing loops can occur. Fig. 3.3 illustrates this routing loop creation for the network of Figure 3.1, assuming link $D \rightarrow F$ fails while node C is restarting.

To prevent such routing loops, (Moy et al., 2003) takes a conservative approach and recommends to revert to a normal OSPF restart when a change in the network topology occurs. However, not every topological change will result into a routing loop even if the restarting router is unable to adjust its forwarding table. This observation underlines the need for a solution able to detect beforehand the creation of loops while



(a) before failure of $D \to F$ (b) after failure of $D \to F$

Figure 3.3: Destination graphs to F before and after failure of link $D \to F$.

a graceful restart is in progress, and possibly to temporarily fix them, in order to avoid the burden of several complete OSPF reconvergences and possible perturbations in the load balancing.

3.2 Properties of routing graphs

Definition 1 *The topological graph of an OSPF network is a weighted directed graph $G = (V, E, w)$ where the finite set V denotes vertices (or 'nodes' or 'routers'), $E \subseteq V \times V \setminus \{(v, v),\ v \in V\}$ denotes the arcs (or 'links'), and $w : E \to \mathbb{R}^+$ is the weight (or cost) function on links. It is assumed that any node is reachable from any other in G (see below).*

Definition 2 *A path from u to v in $G = (V, E, w)$ is a sequence of vertices $p = (v_0, v_1, ..., v_n)$ of V such that $v_0 = u$, $v_n = v$, and each $(v_i, v_{i+1}) \in E$ for $0 \leq i < n$. When such a path p exists from u to v, v is said to be reachable from u through p, denoted $u \overset{p}{\rightsquigarrow} v$ (or simply $u \rightsquigarrow v$). v is called descendant of u and u is called ancestor of v. A circuit in G is a path such that $u = v$. The weight/cost of path p is the sum of the weights/costs of its arcs: $w(p) = \sum_{i=1}^{n} w(v_{i-1}, v_i)$. The distance between u and v is then $d(u, v) = min\{w(p) : u \overset{p}{\rightsquigarrow} v\}$, and a shortest path between u and v is a path reaching this bound.*

We use two types of routing graphs in the sequel, source and destination graphs, attached to any node of $G = (V, E, w)$.

Definition 3 *A source graph $H^\sigma = (V, E')$ is a directed graph such that every node has a unique predecessor for E', except a unique node $\sigma \in V$ (the source), which has none: $\forall v \in V \setminus \{\sigma\}$, $|\{u : (u, v) \in E'\}| = 1$ (and 0 for $v = \sigma$). H^σ is said to be correct iff it contains no circuit. In $G = (V, E, w)$, the source graph $G^{\sigma *} = (V, E^{\sigma *})$ of a node $\sigma \in V$ is obtained by gathering consistent shortest paths from σ to all other nodes in G: $E^{\sigma *} \subseteq E$, and $\forall v \in V \setminus \{\sigma\}$ the unique path p such that $\sigma \overset{p}{\rightsquigarrow} v$ in $G^{\sigma *}$ satisfies $w(p) = d(\sigma, v)$ in G.*

Definition 4 *A destination graph $H_\delta = (V, E')$ is a directed graph such that every node has a unique successor for E', except a unique node $\delta \in V$ (the destination), which has none: $\forall u \in V \setminus \{\delta\}$, $|\{v : (u, v) \in E'\}| = 1$ (and 0 for $u = \delta$). H_δ is said to be* correct

iff it contains no circuit. In $G = (V, E, w)$, the destination graph $G^*_\delta = (V, E^*_\delta)$ of a node $\delta \in V$ is obtained by gathering consistent shortest paths to δ from all other nodes in G: $E^*_\delta \subseteq E$, and $\forall u \in V \setminus \{\delta\}$ the unique path p such that $u \overset{p}{\rightsquigarrow} \delta$ in G^*_δ satisfies $w(p) = d(u, \delta)$ in G.

Source and destination graphs naturally appear in OSPF: by connecting the forwarding rules to destination δ in all nodes of topology G, one gets a destination graph G_δ. Ideally, each G_δ coincides with the true G^*_δ if all nodes have an accurate and up-to-date knowledge about G. However, during a graceful restart, the G_δ in use may differ from the expected G^*_δ, due to frozen forwarding tables, and thus may contain circuits. Similarly, one could build the source graphs G^σ actually used by OSPF for topology G: for each source σ, u is the unique predecessor of v if a packet originating from σ and addressed to v reaches it through u. Ideally again, G^σ should coincide with $G^{\sigma *}$, but this may not hold during a graceful restart. Observe that source and destination graphs are dual notions: inverting the orientation of edges in a source graph yields a destination graph.

As destination graphs encode the effective forwarding rules applied by OSPF, they are instrumental in the prediction of routing loops. These simple objects have numerous properties that can help for this task.

Definition 5 *In a directed graph $G = (V, E)$, the 'connected to' relation on vertices, denoted by $u \sim v$, is defined as the equivalence relation on V generated by $u \rightsquigarrow v \Rightarrow u \sim v$ (this amounts to dropping the orientation of edges). A* connected component *of G is a subgraph $G_{|V'} = (V', E_{|V' \times V'})$ of G such that $V' \subseteq V$ is an equivalence class of vertices for $\sim$.*

Proposition 1 *Let H_δ be a destination graph, each connected component of H_δ either contains δ or contains a unique circuit. Therefore, if H_δ contains p circuits, then it contains $p + 1$ connected components.*

Proof: If a connected component contains two circuits, there must exist a 'path' relating these two circuits, where the notion of path here ignores the direction of edges. Then necessarily there exists a node on this path that has two successors, which violates the definition of a destination graph. Then observe that in a connected component containing a circuit, the number of edges is equal to the number of vertices. By definition, the number of edges in H_δ is equal to the number of vertices minus 1. So there is exactly one connected component in H_δ with no circuit at all. And it must contain a vertex that has no successor, which can only be δ. □

As an example, the destination graph G_F in Figure 3.3(b) contains two connected components. All destination graphs have a similar shape, with connected components made of a single circuit and directed trees descending towards it, plus one last tree directed toward the destination node.

Corollary 1 *Let $G^*_\delta = (V, E^*_\delta)$ be the destination graph gathering the shortest paths to δ in $G = (V, E, w)$. Let G_δ be a perturbed version of G^*_δ where k nodes have modified their successor. Then G_δ contains at most k circuits, and $k+1$ connected components.*

The perturbations above model the fact that k routers are not using the forwarding table they should follow on topology G, but rely on an outdated one. As a consequence, if the topology changes while k routers are operating a graceful restart, at most k routing loops can be created for each destination δ. And as shown in Section 3.5, the same loop can alter several destinations at a time. This suggests that few 'problems' should actually appear and require fixing.

3.3 Prediction of routing loops

Let $G_0 = (V, E_0, w_0)$ denote the topology with which a restarting router r computed its last forwarding table, and let G^r be the source graph of node r in this topology. In G^r, node r has $h_1, ..., h_K$ as successors, which are also helper nodes by design of the graceful restart procedure. Let $\mathcal{D}_r(h_k) = \{v : h_k \rightsquigarrow v \text{ in } G^r\}$ denote the descendants of h_k in G^r, $1 \leq k \leq K$. As G^r is a correct source graph, the $\mathcal{D}_r(h_k) \cup \{h_k\}$ form a partition of $V \setminus \{r\}$.

Let $G_1 = (V, E_1, w_1)$ denote the actual network topology. We assume that the links $(r, h_k) \in E_0$ are still present in E_1. Let G^{h_k} be the source graph of node h_k in this new topology, for $1 \leq k \leq K$ (Figure 3.4). Let $\mathcal{D}_{h_k}(r) = \{v : r \rightsquigarrow v \text{ in } G^{h_k}\}$ denote the descendants of r in G^{h_k} ($\mathcal{D}_{h_k}(r)$ contains u and all nodes below u in Figure 3.4).

Finally, for a node $\delta \in V$, let G_δ be the actual destination graph to δ when all nodes use topology G_1 except r that uses topology G_0.

Proposition 2 *There exists a (unique) routing loop in the destination graph G_δ iff there exists a (unique) k such that $\delta \in \mathcal{D}_r(h_k) \cap \mathcal{D}_{h_k}(r)$.*

Proof: If $\delta \in \mathcal{D}_r(h_k) \cap \mathcal{D}_{h_k}(r)$, then node r will forward a message addressed to δ to its neighbor h_k. And similarly, h_k will direct this message on a path that meets r before δ, whence the creation of a routing loop, containing both r and its successor h_k.

Conversely, a routing loop in G_δ must go through r, the only node performing an inappropriate routing. A packet addressed to δ will leave r through some h_k, so $\delta \in \mathcal{D}_r(h_k)$, and h_k is also on the rooting loop. Since the packet will ultimately come back to r before reaching its destination, one must have that r is on the unique path from h_k to δ in G^{h_k}, so $\delta \in \mathcal{D}_{h_k}(r)$.

The unicity of the routing loop, when it exists, comes from Corollary 1. And the unicity of the helper node h_k revealing this loop comes from the fact that the $\mathcal{D}_r(h_k)$ are disjoint. □

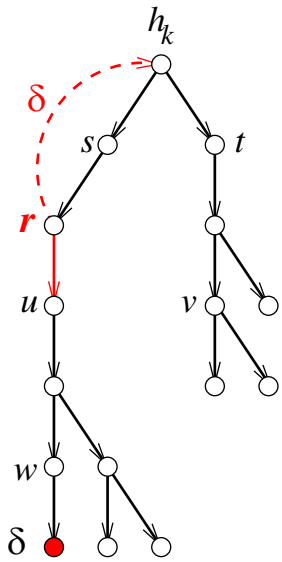


Figure 3.4: The expected source graph G^{h_k} of h_k, and its misbehavior for packets addressed to δ. Instead of correctly forwarding such packets to u, node r selects a wrong neighbor and actually send them back to h_k, thus creating a circuit.

This defines a simple practical test for discovering destinations δ at risk, i.e. unreachable due to the presence of a routing loop. All neighbors of r are advertised that r initiates a graceful restart, and they act as helpers, so they can store the frozen G^r used by r all along the grace period. Each successor h_k of r can then determine and announce the contents of $\mathcal{D}_r(h_k) \cap \mathcal{D}_{h_k}(r)$ if the topology changes.

3.4 Correction of a routing loop

Let $G_\delta = (V, E_\delta)$ be a destination graph over topology $G = (V, E, w)$, so $E_\delta \subseteq E$. If G_δ is correct (i.e. contains no circuit), it can reliably be used to forward packets to δ, but it may not use the shortest paths of G. Assume that k routers $r_1, ..., r_k \in V$ are performing a graceful restart in G. As seen above, the actual G_δ used for forwarding packets to δ differs from the optimal G^*_δ by at most k arcs: the arcs originating from routers $r_1, ..., r_k$ (assuming they differ from δ) can point to any node in V. Therefore G_δ contains at most k circuits, that each contain at least one node of $\{r_1, ..., r_k\}$. Given that these nodes cannot change their forwarding rule, is it possible to modify the routing choices of other nodes to turn G_δ into a correct destination graph ? What is the minimal number of nodes that should be rerouted, and where are they ?

3.4.1 Severity Degree of Routing Loops

Definition 6 *Let $G_\delta = (V, E_\delta)$ and $G'_\delta = (V, E'_\delta)$ be two destination graphs in G such that r has the same successor in G_δ and in G'_δ. Let us denote by $C(G_\delta, G'_\delta) = |E_\delta \backslash E'_\delta| =$*

$|E'_\delta \setminus E_\delta|$ the number of arcs that distinguish them. The color of node v in G_δ is defined as $C_\delta(v) = min\{C(G_\delta, G'_\delta) :\ v \rightsquigarrow \delta$ in $G'_\delta\}$.

So $C_\delta(v)$ is the minimal number of reroutings that should take place in G_δ in order to correctly forward packets from v to destination δ. Note that $C_\delta(v)$ can be infinite if no correction is possible, and $C_\delta(v) = 0$ iff v is in the connected component of G_δ that contains δ.

Proposition 3 *Let G_δ be a destination graph in topology G. If $u \rightsquigarrow v$ in G_δ, then $C_\delta(u) \leq C_\delta(v)$. And if the arc (u, v) exists in G, then $C_\delta(u) \leq C_\delta(v) + 1$.*

Proof: Assume $u \overset{p}{\rightsquigarrow} v$ in G_δ (path p is unique). Let G'_δ be obtained by rerouting $C_\delta(v)$ vertices of G_δ, in such a way that $v \overset{p'}{\rightsquigarrow} \delta$ in G'_δ (again path p' is unique in G'_δ). If no vertex of path p has been rerouted, then path p still exists in G'_δ, and connecting it to p' yields a path from u to δ, so $C_\delta(u) \leq C_\delta(v)$. Otherwise, some nodes of path p have been rerouted in G'_δ, so p does not exist anymore in G'_δ. This means that path p' uses nodes of path p, so the path originated at u in G'_δ goes to δ, and $C_\delta(u) \leq C_\delta(v)$.

For the second part, consider G'_δ defined above, where $v \overset{p'}{\rightsquigarrow} \delta$. If path p' does not go through vertex u, let G''_δ be obtained by rerouting u to v in G'_δ. Then $u \rightsquigarrow \delta$ in G''_δ, so $C_\delta(u) \leq C_\delta(v) + 1$. If path p' goes through u, then $u \rightsquigarrow \delta$ in G'_δ, so $C_\delta(u) \leq C_\delta(v)$. □

As a consequence, the color of nodes in each connected component of G_δ augments as one progresses towards the circuit, and it is constant on this circuit. There cannot be gaps in series of colors: vertices of color n exist only if there exist vertices of color $n - 1$.

Corollary 2 *G_δ contains a circuit which color is infinite iff this routing loop cannot be corrected. The color of a circuit in G_δ is the number of reroutings that is necessary to redirect to δ all nodes of the connected component containing this circuit. If G_δ contains a unique circuit, its color is the minimal (and sufficient) number of reroutings to transform G_δ into a correct destination graph.*

Figure 3.5 illustrates the vertex coloring on the destination graph G_F, for our running example. Vertices E, F, G are located in the same connected component as the destination F, therefore their color is 0 (displayed in green). One has $C_F(A) = 1$ (yellow), because edge (A, E) exists in topology G, and $C_F(E) = 0$. A can easily reach F by rerouting packets through E instead of B in G_F. Finally, vertices B, C, D in the circuit all have color 2 (red). C is the frozen restarting router, so it cannot be rerouted, and neither B nor D could be directly rerouted to E, F or G (recall that link (D, F) failed). However, B can be rerouted to A, and the latter to E. These two modifications are sufficient to guarantee that all packets addressed to F actually reach it.

Proposition 3 reveals a simple *coloring algorithm* over G_δ. Nodes of color 0 are easily obtained by back-tracking from δ. For any remaining (uncolored) node u, if arc (u, v) exists in G and v has color 0, then u takes color 1. And one can recover all nodes

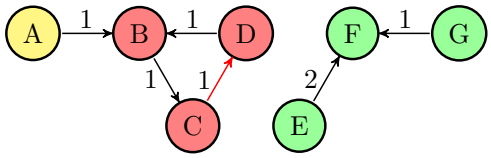


Figure 3.5: Vertex colors on the destination graph G_F: green=0, yellow=1, red=2.

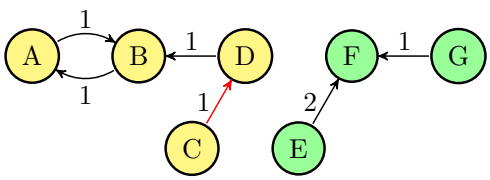
of color 1 on G_δ by backtracking from such u nodes. Similarly, nodes of color 2 are the uncolored predecessors u in G of a node v of color 1, or the uncolored ancestors in G_δ of such u nodes. And so on, until no more coloring rule is applicable. The remaining uncolored nodes take ∞ as color. This algorithm has a linear complexity, similar to Dijkstra's algorithm, and it can also be distributed. It allows one to decide if routing loops can be corrected.

3.4.2 Correction of a Routing Loop

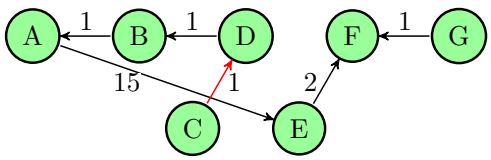
The remainder of the chapter focuses on the case of a single restarting router in G. Therefore, if destination graph G_δ is incorrect, there is a single routing loop to repair.

Corollary 3 *Let the incorrect destination graph G_δ contain a unique circuit p of color n. At least one node of this circuit (different from the frozen node r) can be rerouted to a node of color $n-1$. Performing this rerouting yields the destination graph G'_δ that again contains a unique circuit p', of color $n-1$.*

This result derives simply again from Proposition 3. Its interest is to reveal a simple procedure to determine the n reroutings that can turn G_δ into a correct destination graph. Figure 3.6 illustrates these two steps for the G_F in Figure 3.5: B is first rerouted from C to A, then A is rerouted from B to E.



(a) New vertex colors once B is rerouted from C to A.



(b) New vertex colors once A is also rerouted from B to E.

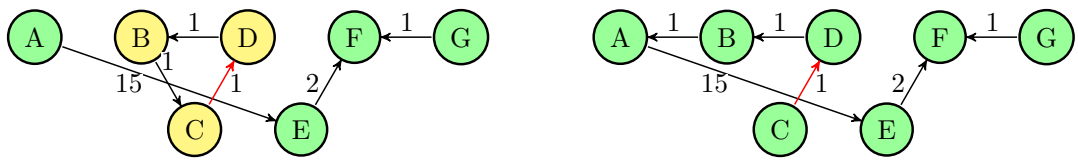
Figure 3.6: Successive reroutings to correct destination graph G_F.

In summary, when a topological change occurring during a (single) graceful restart creates a routing loop for some destination δ, a simple procedure can determine the minimal number n of reroutings that could correct it, and the location of these reroutings. There generally exist several such temporary 'patches' of n reroutings, and one could wonder which one is the most efficient in terms of average cost, if link weights are taken into account. We conjecture that this problem is NP hard. One may wonder

about situations where the color of the circuit is infinite. In that case, there is no solution to reroute messages to δ around r. Therefore a standard restart of OSPF would also be useless to resolve the problem.

3.4.3 Scheduling of Backup Routings

Assume one has determined a sequence $s_1, ..., s_n$ of vertices that should be rerouted to correct a destination graph G_δ, where the index i in s_i represents the color $C_\delta(s_i)$. In which ordering should these temporary reroutings be performed ? One possibility is illustrated in Figure 3.6, where $s_2 = B$ is rerouted before $s_1 = A$ in G_F. The reverse order is illustrated in Figure 3.7. As one can notice, this second option offers a better transient mode: nodes are progressively rerouted correctly to $\delta = F$, whereas in the previous option all nodes suffer from the loop until the last rerouting is performed.



(a) New vertex colors in G_F once A is rerouted from B to E.

(b) New vertex colors in G_F once B is then rerouted from C to A.

Figure 3.7: Successive reroutings to correct destination graph G_F.

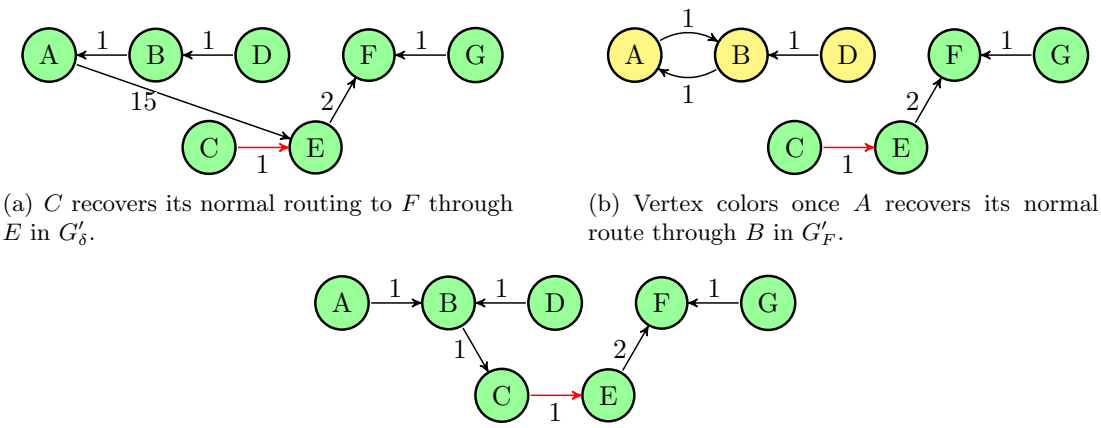
Proposition 4 *Let $s_1, ..., s_n$ be a minimal sequence of vertices that should be rerouted to correct G_δ, where $C_\delta(s_i) = i$ in G_δ, and $n = C_\delta(r)$. Rerouting only s_i to its appropriate new successor yields G'_δ where the new node colors satisfy $C'_\delta(s) = C_\delta(s)$ if $C_\delta(s) < i$, and $C'_\delta(s) = C_\delta(s) - 1$ if $C_\delta(s) \geq i$.*

Proof: Each s_i will be rerouted toward a vertex of color $i-1$ in G_δ, and in particular to an ancestor of s_{i-1}. Performing this rerouting decreases by one the color of s_i in G'_δ, since it is now a predecessor of a node of color $i-1$. Similarly, all nodes that were at distance $j \geq i$ now have one less rerouting to perform to reach δ. The rerouting of s_i cannot help node of color strictly lower than i, since by definition of the color they have a shorter path to δ in number of reroutings. □

A consequence of this result is that one should start rerouting nodes in the order $s_1, ..., s_n$, in order to maximize the color decrease in G_δ, i.e. to maximize at each step the number of nodes that can correctly reach δ.

Assume now that the restarting router r has finished its graceful restart. Can it safely switch to its new forwarding table (corresponding to the actual topology G_1) ? And how should one remove the temporary rerouting patches ? Figure 3.8 illustrates the return in function of $r = C$, now correctly connected to E, and a removal of the rerouting patches following order $s_1 = A, s_2 = B$. As one can notice, this may

recreate forwarding loops, whereas the converse ordering is safe. A similar phenemenon was already observed in standard OSPF convergence, and led to the developement of ordered updates of forwarding tables, known as OFIB (Francois and Bonaventure, 2005, 2007; Hock et al., 2011).



(a) C recovers its normal routing to F through E in G'_δ.

(b) Vertex colors once A recovers its normal route through B in G'_F.

(c) Vertex colors once B then recovers its normal route through C in G'_F.

Figure 3.8: Successive removals of the temporary reroutings into the corrected destination graph G'_F, after node C returns to function.

Proposition 5 *Let $s_1, ..., s_n$ be a minimal sequence of vertices that have been rerouted to correct the loop created by r in G_δ, where $C_\delta(s_i) = i$ in G_δ, and $n = C_\delta(r)$. Once r returns to function, it can safely switch to its expected forwarding table without recreating a loop. And removing the temporary reroutings starting from s_n to finish by s_1 guarantees that no transient routing loop appears.*

Proof: Consider G_δ, before rerouting $s_1, ..., s_n$, and G'_δ its corrected version after $s_1, ..., s_n$ have been rerouted. Before correction, G_δ contains a unique circuit, due to the erroneous routing of r, and nodes $s_1, ..., s_n$ are ancestors of this circuit. The second connected component of G_δ is a directed tree where each node has a path to δ. Rerouting r to its correct (expected) neighbor for topology G_1 and destination δ would turn G_δ into the correct and expected destination tree to δ for topology G_1. Therefore the correct successor of r cannot be in the connected component containing the circuit (otherwise packets would come back to r), and is necessarily in the connected component containing δ. Performing this rerouting of r in G'_δ is thus harmless: it simply establishes a more direct path to δ for ancestors of r (see Figure 3.8(a), with $r = C, s_1 = A, s_2 = B, \delta = F$).

For the second statement, consider again G_δ, before reroutings of $s_1, ..., s_n$. Assume only nodes $s_1, ..., s_k$, $k < n$ are rerouted. This yields a new destination graph $G'_{\delta,k}$, which remains incorrect because it still contains a loop going through r, but where all node colors have decreased by k, by Prop. 4. In particular, nodes that still have

a strictly positive color in $G'_{\delta,k}$ have r as a descendant, i.e. their outgoing path is directed towards the remaining circuit (See Figure 3.7(a) for an example of $G'_{\delta,1}$, with $r = C, s_1 = A, s_2 = B$ and $\delta = F$). Of course, $G'_{\delta,n}$ coincides with the corrected G'_δ mentioned above. Consider now $G'_{\delta,k}$. If one reroutes r to its correct neighbor for topology G_1 and destination δ, then one gets a new destination graph $G''_{\delta,k}$ where the circuit has vanished (all color nodes go down to 0). To conclude, observe that this correct destination graph $G''_{\delta,k}$ can be obtained by starting from G'_δ, rerouting r to its correct successor, and by reassigning to $s_{k+1}, ..., s_n$ their original successors in G_δ. In other words, removing the rerouting patches starting from s_n and terminating by s_1 does not create routing loops. However, selecting a different order can create loops, as proved by Figure 3.8(b) where the patch at $s_1 = A$ was removed before the one at $s_2 = B$. □

3.5 Correction of multiple routing loops

Assuming a single router r has a frozen forwarding table while the network topology evolves, we have shown how to detect a routing loop for some destination and how to correct it with minimal effort. This leaves open the burden of fixing *all* problematic destinations, which we address now. The idea is that fixing a problematic destination may help resolving others. Consider again the setting of Section 3.3, where all nodes established their forwarding table according to topology G_1 excepted node r, which used topology G_0. We rely on the criterion of Proposition 2.

Proposition 6 *Let δ_1, δ_2 be two destinations in $\mathcal{D}_r(h_k) \cap \mathcal{D}_{h_k}(r)$ where h_k is one of the successors of r in its source graph G^r. Consider the source graph G^{h_k} of node h_k (in topology G_1). If $\delta_1 \rightsquigarrow \delta_2$ in G^{h_k}, then the routing loop to δ_1 and to δ_2 goes through the same nodes. The node reroutings that correct the destination graph G_{δ_1} can be used to correct as well G_{δ_2} (see Figure 3.9).*

Proof: As δ_1, δ_2 are in $\mathcal{D}_r(h_k)$, their packets are forwarded to h_k when they reach r. Consider the unique path relating h_k to δ_2 in the source graph G^{h_k}. This path decomposes as the concatenation of three segments $p_0 \cdot p_1 \cdot p_2$ where path p_0 relates h_k to r, path p_1 relates r to δ_1, and path p_2 relates δ_1 to δ_2. The path $p_0 \cdot p_1$ from h_k to δ_1 goes through r, so the (unique) routing loop to δ_1 in the destination graph G_{δ_1} to δ_1 is defined by arc (r, h_k) followed by path p_0. It is thus the same routing loop as the one appearing in G_{δ_2} to δ_2.

To correct the routing loop to δ_1, one must reroute at least one node on segment p_0 (excepted r itself), and possibly other nodes elsewhere in G_{δ_1}. After this correction (see Section 3.4), $C_{\delta_1}(r)$ nodes have changed their successor for destination δ_1. Let G'_{δ_1} be the corrected destination graph to δ_1, and $\bar{p}$ the new path from h_k to δ_1 in G'_{δ_1}. Two situations can occur. If $\bar{p}_1$ uses no node of path p_2, then no node along path p_2 changed their rerouting to help h_k reach δ_1, therefore in the new source graph of h_k, ${G^{h_k}}'$, path p_2 relating δ_1 to δ_2 of G^{h_k} has been preserved. Therefore the concatenation of paths $\bar{p}_1 \cdot p_2$ relates h_k to δ_2. In other words, if the $C_{\delta_1}(r)$ nodes that changed their forwarding

rule for destination δ_1 adopt the same rule for destination δ_2, the forwarding circuit to δ_2 is repaired. In the second case, path $\bar{p}_1$ does use some node(s) of path p_2 relating δ_1 to δ_2 in G^{h_k}. The remark above still holds: packets addressed to δ_2 can follow the same modified path $\bar{p}_1$ as those addressed to δ_1, until they reach a node of path p_2. In that case they can safely follow the rest of this path down to δ_2. □

This also proves that the color $C_{\delta_2}(r)$ of the routing loop (to δ_2) in G_{δ_2} is lower than the color $C_{\delta_1}(r)$ of the routing loop (to δ_1) in G_{δ_1}. But as illustrated by the second case discussed above, it can be strictly lower.

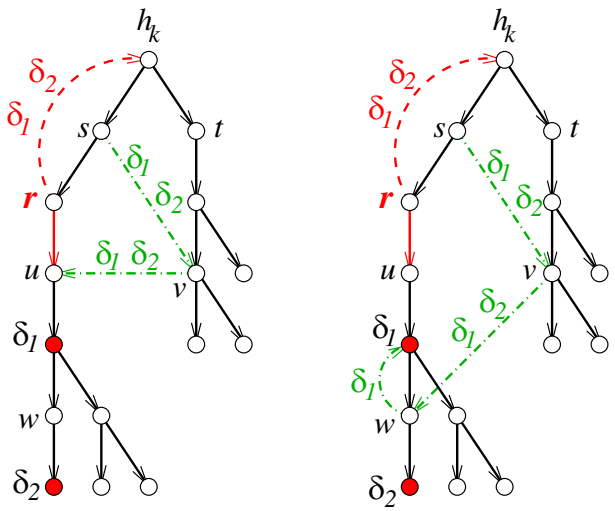
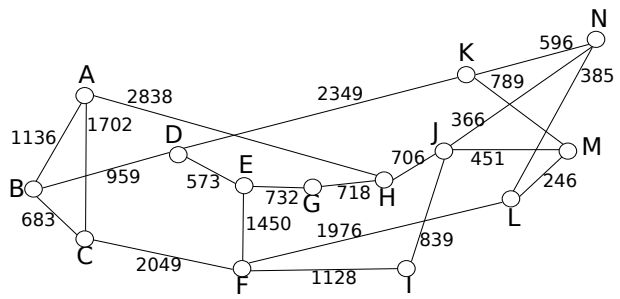


Figure 3.9: Rerouting packets addressed to δ_1 with a minimal number of hops in order to go around r (and thus avoid the circuit). This rerouting path can also be (partly) used to correct the circuit on the path to δ_2 when $\delta_1 \rightsquigarrow \delta_2$ in G^{H_k}.

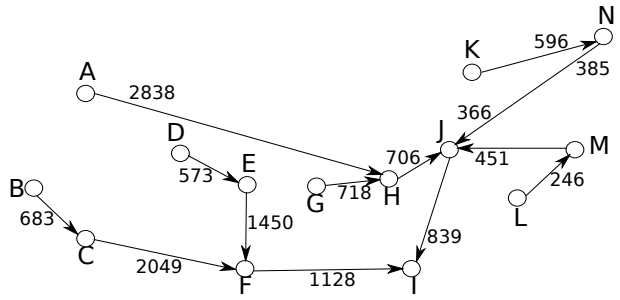
3.6 Evaluation of the enhanced graceful restart

To illustrate the potential gains of the proposed enhanced graceful restart, we consider the NSFNET (Figure 3.10(a)), a US network based on a former NSF network topology used in many studies, e.g. (Hülsermann et al., 2004).

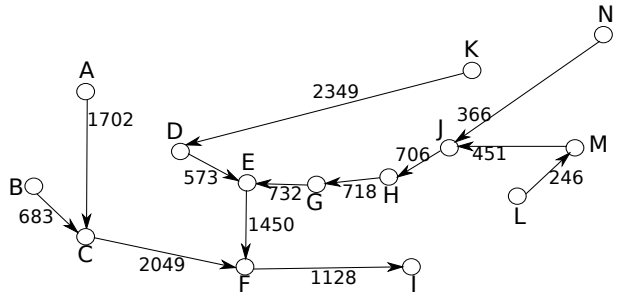
In the destination graph G_I (Figure 3.10(b)), router L is supposed to be restarting and thus has a frozen forwarding table. If any link in $\{A-B, A-C, B-D, D-K, E-G, K-M, F-L, N-L\}$ fails, no routing loop will occur for destination I if L keeps its frozen routing table instead of adopting the new one expected from it. Therefore, removing L from the forwarding path as it is recommended by the standardised graceful restart is unnecessary. L can safely update its routing table (towards destination I) after it completes its restart and re-establishes adjacency with its neighbors. By contrast, graceful restart is pessimistic and demands to advertise the disconnection of L, and later to announce its return in the topology, which incurs an extra round of flooding, routing



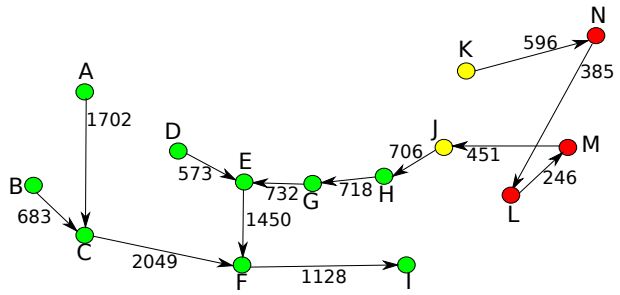
(a) 14-node NSFNET topology (link distances in km).



(b) Destination graph G_I.



(c) New destination graph G'_I, loop avoided using standardised GR.



(d) New destination graph G''_I, loop avoided using enhanced GR. Colors are 0=green, 1=yellow, 2=red.

Figure 3.10: Evaluation on the NSFNET network

table calculations and forwarding table updates, besides some unnecessary temporary reroutings. Our proposal can avoid this second round by detecting that no routing loop is about to occur, even if the rebooting router does not behave exactly as expected for some short period of time.

Now suppose that link $I - J$ fails while router L is restarting. The new shortest paths to destination I require that nodes J, K, M, N route their packets through L, the latter being expected to forward them to F. Keeping L in the topology with its frozen routing table would create the loop $M \to L \to M$. The standardized graceful restart avoids this by removing L from the topology, which results in the destination graph G'_I (Figure 3.10(c)). Observe that J routes its traffic to I through H instead of M and L; and K routes its traffic through D instead of N and L. Once L completes its restart, J, K, M, N will reorient their traffic for I through L. This represents in total six modifications in their routing tables.

With our proposal, the loop $M \to L \to M$ is detected and temporarily patched, resulting in destination graph G''_I (Figure 3.10(d)). Observe that routers K and N are directly set to their correct final routing. Only M and J are temporarily rerouted to patch the routing loop. Once L returns in function and updates its table, M, J can safely adopt their final value. This represents a total of four modifications in the routing tables of J, M, N, since two of them are directly positioned to their final value.

3.7 Discussion

This work shows that it is possible, at low complexity, to preserve the graceful restart procedure of OSPF routers even if the topology changes during this operation. To this end, the helper nodes of the rebooting router simply have to check if routing loops will appear, and in that case to compute the optimal patches (temporary reroutings) for all problematic destinations. They then ask the selected nodes to apply these patches, and later to remove them when the rebooting router is back, all this in an appropriate ordering. Helper nodes are also in charge of moving from one set of temporary patches to another set, in case the topology evolves again during the reboot. This is thus a minimal extension to the existing graceful restart standard, which incurs smoother traffic perturbations since no massive rerouting is involved to bypass the potentially dangerous router. These ideas extend to several simultaneous graceful restart operations: n frozen routers can cause at most n loops toward some destination. However, patching optimally these loops will require the coordination of the n sets of helper nodes. This will be examined in a forthcoming work, together with an extensive evaluation of this enhanced graceful restart.

Modern IP networks implement fast corrective mechanisms, as IP Fast ReRoute (IPFRR) (Gjoka et al., 2007), that precompute bypassses for all single link or single node failures, and then rely on ordered updates of forwarding tables (OFIB) to move to the new routing rules computed by OSPF (Francois and Bonaventure, 2005, 2007; Hock et al., 2011). These fast protection ideas are of course compatible with the work presented here, provided their computations take into account the frozen routing table

of a rebooting router, and the patches that have been applied. Notice however that they serve a different purpose since their scope is to quickly and harmlessly isolate a faulty or dead element, while an enhanced graceful restart aims specifically at maximally exploiting a not yet dead element, despite its non optimal behavior.

Chapter 4

State of the art

Contents

4.1 Rule-based Expert systems **46**
4.2 Model-based Expert systems **51**
4.3 Case-based Reasoning systems **54**
4.4 Knowledge discovery and data mining **56**
4.5 Fault-symptom graphs and the codebook approach **64**
4.6 Dependency/causality graphs **68**
4.7 Chronicles: patterns of timed events **73**
4.8 Bayesian Networks **78**
4.9 Summary **87**

The last decade has witnessed a spectacular development of telecommunication networks. They are no longer distinguished from computer networks, and connect a huge variety of equipment, ranging from "smart" wireless mobile devices to data centers. They also support increasingly diverse services and applications, ranging from real time communications (voice, streaming, conferencing, gaming) to data storage and access (social networks, cloud services, online businesses) and to intensive distributed/networked applications.

This evolution has impacted network structures in their size, complexity and heterogeneity, while the demand for availability, reliability and quality of service was simultaneously becoming prominent. Network management is thus becoming a major concern for operators, which has triggered intensive research about *autonomic networking,* the network counterpart of the concept of autonomic computing advocated by IBM in 2001. Autonomic networking aims at designing networks where most of the classical management operations would be automatized as much as possible, in order to program the network by high-level objectives or policies, and let the network implement the best answer to these requirements. One objective being to relieve operators from numerous tedious and error prone micro-management operations, and at the same time to shorten the time to market of new services. Autonomic networking is also called self-management, and decomposes the classical FCAPS management functions (faults, configuration, accounting, performance, security) into a collection of self-* func-

tions, going beyond these 5 historical objectives: self-configuration, self-optimization, self-diagnosis, self-healing, etc.

This thesis focuses on the management of faults, and thus aims at contributing to the design of self-diagnosis and self-healing methods. Besides, it advocates the new concept of self-modelling: the ability to automatically build an abstract model of the network that will be the support of diagnosis and healing algorithms.

Faults, also called failures, root causes, or primary causes, represent malfunctions in network equipment or software that have an impact on the expected service(s) from this network. Given the very design of networks as a hierarchical assembling of interdependent functions and components, faults naturally *propagate*, i.e. generate a cascade of secondary failures or malfunctions. For example a broken link cuts an end to end communication; a missing message can make a protocol fail an important step, that in turn may prevent the establishment of a connection, whence the impossibility to use a service, etc. Network equipment, protocols and services are equipped with elementary monitoring indicators that check their health and raise *alarms* when a malfunction or a breakdown is detected. Alarms are primarily directed to the management layer for display in dashboards, from which the operator will decide which mitigation, recovery or maintenance operation is necessary. Some alarms have also an operational role: they can be directed to client functions in the network, for example to automatically switch protection mechanisms (as the automatic rerouting in case of link failure), or simply for a simple informative purpose of that client, which may itself trigger a new alarm. The consequence of both fault and alarm propagations is that a single root cause may result in a complex and distributed pattern of subsequent failures and of their corresponding alarms.

The first expectation from a fault management system is the ability to detect (Bouloutas et al., 1994) the occurrence of primary faults, and then to localize/isolate them (Bouloutas et al., 1995; Katker and Geihs, 1997). This consists in processing the various alarms or malfunction indicators (such as customer complaints) in order to identify the possible root cause(s). This task is sometimes referred to as fault localization, alarm/event correlation, or root cause analysis. Event correlation was primarily performed by human operators, in response to customer complaints, before mitigation or corrective actions could be envisioned. The current practice in management rooms has long been to ignore the numerous non severe alarms, as they may result from non advertised maintenance operations on the network, or of temporary reconfigurations, with little impact on the users. But even for the few selected cases, fault management was soon identified as too complex for humans, who can keep track of only few hypotheses in their reasoning, and who need a long training to fully master their network segment. Besides, the requirements of fault management have been considerably reinforced: it should now

- capture several layers and segments of the network at a time, and correlate events/alarms horizontally (within the same layer, but over several network segments) but also vertically (cross-layer correlation),
- be tailored to a specific network instance, and adaptive as the network architecture/configuration evolves,

- be not only reactive, to deal with faults that have already occurred, but also be proactive, i.e. early detect failures that yet do not have serious consequences, but which may degenerate,
- classify accurately the different malfunctions, and evaluate their impact on the services (with a specific focus on the services to customers),
- provide the human operator with clear explanations about what (can have) happened,
- suggest corrective/repair actions, and possibly perform them automatically (with or without a human in the loop), with explanation/verification of their expected effect,
- extend beyond a passive procedure reasoning on the collected alarms, and become an active procedure able to poll components or perform tests in order to collect the most relevant information for the current state of the reasoning process,
- be designed as a distributed procedure, to coordinate the management teams in charge of different network segments or layers within the same operator, but also extend to multi-operator cooperation to address cross-domain malfunctions (which immediately triggers confidentiality issues).

Besides these strong expectations, fault management is facing difficulties that are specific to the networking context. They mostly relate to a) the quality of the information that is collected, b) the difficulty to reason about a very complex system. Considering the alarm side first, one is far from the familiar setting where a dynamical system produces a sequence of observable events, as in hidden Markov models (HMMs) for example. Several new phenomena must be captured, such as

- the multiplication of alarms, due to repetitive failed attempts to use a service, or the intermittent/repetitive alarms (for example those reacting to threshold crossings);
- missing alarms, due to their loss (for example in case of inband signalling) or due to their masking (incomplete propagation of faults), to their filtering by some intermediate equipment, to the fact that the supervisor did not register to some type of alarms, or to their too late arrival;
- the inconsistency of alarms, due to different perceptions of the state of some network resource;
- ambiguous alarms, when several underlying phenomena are loosely gathered into a single type of default indication (this is frequent when low-level failure indicators are reported to an upper layer);
- transient alarms due to temporary states of the network, during reconfiguration or maintenance operations;

- delayed alarms, reordered alarms or more generally the fact that a single failure generates alarms at different locations in the network, which results in a partially ordered pattern of alarms rather than a nice sequence,
- the interferences and/or interleaving of alarm patterns in case of multiple primary failures;
- the necessity to take the timestamps of alarms into account;
- the difficulty to define the relevant range of alarms to consider among the numerous types raised by the different network components.

This last item introduces the second difficulty: how to reason about alarms in such large complex systems ? This immediately raises several questions:

1. What type of knowledge should one use to support his reasoning?
2. How should this knowledge be modeled or formalized?
3. Which centralized and/or distributed algorithms can exploit this knowledge, at least to recover primary failures from alarms in the first place?

Surprisingly, while alarm processing clearly falls within the scope of automatics, the complexity of the problem has rather attracted the artificial intelligence community. The problem was addressed with numerous techniques in the past two decades, ranging from expert systems that try to mimic human expertise, to machine learning techniques (neural networks, statistical inference, case-based reasoning), and to model based reasoning.

The rest of this chapter reviews some of the numerous contributions to the topic of fault and alarm management. It does not aim at exhaustivity, but rather tries to sample the domain in order to give an overview of the techniques that were proposed and experimented, before discussing their advantages, drawbacks and positioning the ambition of this thesis. For an excellent survey of various fault localization techniques and algorithms the reader is referred to (Steinder and Sethi, 2004a).

4.1 Rule-based Expert systems

The application of expert systems in network operations and management software development has been a growing phenomenon (Cronk et al., 1998). Diagnostic expert systems attempt to infer the cause of a problem from symptoms recognized in sensor data. An expert system is a problem-solving software that embodies specialized knowledge in a narrow task domain to do work usually performed by a trained, skilled human. Expert knowledge in the task domain must lend itself to a formalized representation to be implemented in a knowledge base. Various types of knowledge representation schemes can be employed: rules, frames, semantic networks, lists of facts, logic predicates, etc. The complexity of diagnostic systems make knowledge representation a critical issue—see (PAU, 1986) for an evaluation of knowledge representation

schemes. However, an expert system will often require a combination of the previous types. With regard to the inference mechanism for expert systems, the various options include: forward-backward chaining (still the most common), generate/test methods, heuristic search, and meta-rules (PAU, 1986). As with the knowledge representation, a combination of the above inference mechanisms is common.

Surveys of the first diagnostic expert systems of technological processes are provided by (PAU, 1986) and (Scherer and White, 1987). The first successful diagnostic expert systems (also called "first generation" expert systems) were rule-based and used empirical reasoning. Since then, numerous systems have been built. Expert systems are organized around three levels: data, control, and task knowledge. Depending upon the type of rule-based system, these three levels become, architecturally: working memory or global database (data), knowledge base (task knowledge), and inference engine (control) (Cronk et al., 1998). This architecture is shown in Figure 4.1. At the control

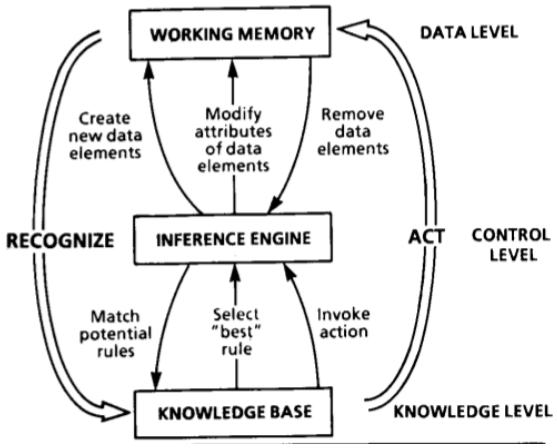


Figure 4.1: Organization of rule-based expert systems, taken from (Cronk et al., 1998)

level, an inference engine is a program which determines how to apply knowledge contained in a knowledge base to current facts and premises described in working memory in order to infer new data, which can then be used for further inferences. In a rule-based system environment, the inference engine determines which rules are applicable and which of these candidate rules should be the one to use in a given situation. In rule-based systems, rules are expressed in the form:

```
IF <condition> THEN <action>
```

The inference engine of a rule-based system implements the following concept: before we take an action, we want to consider all possible choices, and make a decision on which choice is best. The control mechanism is a 'recognize-act cycle' (see Figure 4.1). The cycle has the following three components (Cronk et al., 1998):

- *Matching*—finds all the rules that are satisfied by current contents of working memory. These matches are collectively called the 'conflict set'.

- *Selection*, or *conflict resolution*—determines which one of the matches in the conflict set is the best to invoke at this time.
- *Rule invocation* (*execution*)—is the process of applying the matches specified by the chosen rule. These actions typically change data so that new patterns are formed. In some systems, rules themselves are added, removed, or modified.

The cycle is repeated until there are no more matches to invoke (or an explicit halt is issued). Therefore, program control is the repeated evaluation of rule conditions based on changing data rather than on static structure of the program. Thus, the strategy is called data-driven control, or forward-chaining.

Reasoning by the exercising of inference rules can proceed in different ways according to different control procedures. As explained above, the strategy of forward-chaining is to start with a set of facts or given data and to look for rules in the knowledge base the 'IF' portion of which matches the data. When such rules are found, one of them is selected based upon an appropriate conflict resolution criterion and executed or 'fired'. This generates new facts and data in the knowledge base which in turn causes other rules to fire. The reasoning operation stops when no more new rules can fire. It is illustrated in Figure 4.2. An alternative approach is to begin with the goal to be proved

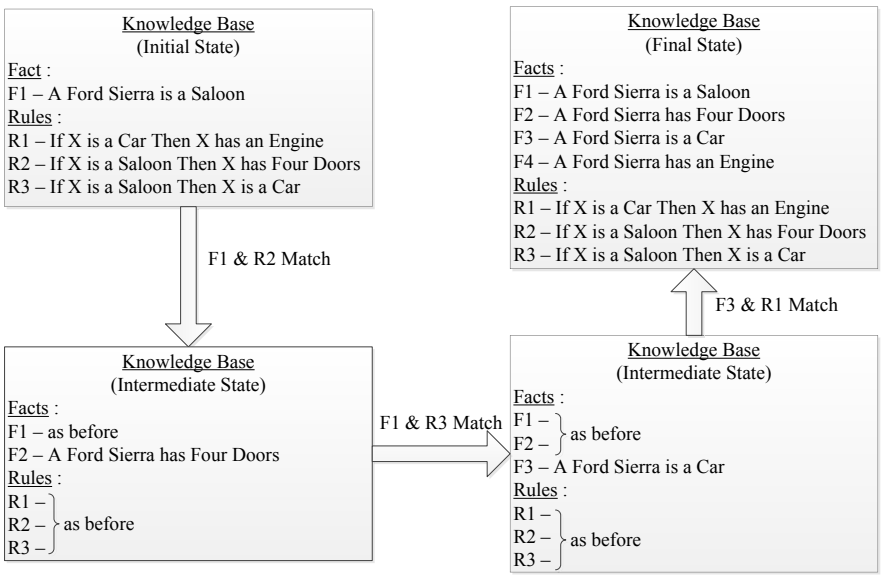


Figure 4.2: An example of forward chaining (taken from (Pham, 1988))

and try to establish the facts needed to prove it by examining rules with the desired goal as the 'THEN' portion. If such facts are not available in the knowledge base, they are set up as sub goals. The process continues until all required facts are found, in which case the original goal is proved, or the situation is reached when one of the sub goals cannot be satisfied, in which case the original goal is disproved. This method of reasoning is called 'backward chaining' or 'goal directed-inferencing'.

Several difficulties are to be taken into account while designing an expert system (David and Krivine, 1987):

1. it is difficult to formalize the problem: what knowledge is useful, how should it be represented and how should it be used;

2. it is difficult to acquire knowledge;

3. it is difficult to validate an expert system.

Furthermore, network troubleshooting presents characteristics like incomplete data, high rate of events, simultaneous presence of several problems, which raise interesting problems in the development of an expert system. (Mathonet et al., 1987) stresses the development issues that are peculiar to network troubleshooting. Of particular importance are performance of inference in real-time, multi-problem handling, and consideration of time in reasoning and revision of belief. Dealing with such issues is primarily a question of system design. This has implications for the knowledge base organization, reasoning mechanism, and recording of deductions. The above challenges and difficulties have been addressed in the literature.

Lor et al. (Lor, 1993) developed an expert system to diagnose multiplexer networks. The diagnostic expertise is classified into general expertise and customized expertise. Their Network Diagnostic Expert System keeps a database containing certain network information, both static and dynamic, needed during the diagnostic process. Information stored include the relationships between the logical entities (channel groups) and the physical entities (nodes or links) like routing information, attributes of the physical entities, and node/link incident relationship, etc. The diagnostic process is bidirectional between the core expertise and the customized expertise. A specific failure invokes a rule in the customized knowledge, which in turn invokes a rule in the generic knowledge. The generic rules offer a general diagnosis and recovery plan for such a failure. The customized expertise is then invoked again to implement this plan. In an ordinary diagnostic session with more than one network faults, the diagnostic system goes back and forth between customized and general expertise many times, at least once for each failure encountered.

Sinergia (Brugnoni et al., 1993) is an expert system for the isolation and diagnosis of faults in the Italian Telecommunications Network. They introduce the notion of 'reduced Fault Influence Area' (RIA) defining the set of equipment in the network affected by a fault. The signals belonging to the RIA are close in time (i.e. occur within few milliseconds). This principle denoted as *temporal locality* is the main theoretical support of the real time diagnostic technique implemented in Sinergia. The overall methodology exploited by Sinergia is built up of two main reasoning steps that implement a sort of generate and test paradigm. The first step is based on a set of rules which instantiate fault hypotheses, while the second is a classical AI heuristic search to determine the best solution among the hypotheses.

DANTES (Mathonet et al., 1987) is an expert system designed to provide real-time assistance to network supervisors in carrying out their troubleshooting activities. It

uses three kinds of knowledge: structural knowledge, deductions, and problem detection/diagnosis knowledge. The structural knowledge uses a structured object formalism comprising two hierarchies: network component and network event. Properties associated with network components serve to represent the object current situation or to represent the relationships between objects. As for properties associated with network events, they serve to identify an event in time and space and to define event treatment characteristics. Deductions can be symptoms, hypotheses or results. Problem detection and diagnosis knowledge specifies how to interpret network events, how to recognize problem situations and how to isolate faulty components. The heuristic nature of this knowledge led naturally to a production rule representation. The rule base in DANTES (Mathonet et al., 1987) is not flat: rules are grouped by object class. This organization allows a distribution of expertise among the different object types of the network representation. DANTES has the ability to deal with time aspects of inference (i.e. time correlation between events and belief revision with time) and present several features which are typical of traditional real-time systems.

ANSWER (Weiss et al., 1998b) is an expert system used by surveillance technicians at AT&T's two network control centres to monitor and maintain the 4ESS switching elements in the AT&T network. In addition to using rule-based programming, ANSWER uses object-oriented technology and models the 4ESS as a collection of devices. The model implicitly contains information about the structure and behavior of the 4ESS. Much of the reasoning in ANSWER is accomplished by using 'affective' relations. Affective relations express aspects of the design at a level of abstraction that expert troubleshooters use to link symptoms to faults, and hence are easily acquired. These relations are maintained and used by rules. The sub-part relation serves to isolate faults. For example, ANSWER has a rule that states that if many of device A's sub-parts fail, then the fault is most likely located in device A (not in its sub-parts). The model of the 4ESS switch is dynamically built from the information sent to it from the 4ESS. The first advantage of such a dynamic model is flexibility: no up-front configuration information is required. The second advantage is efficiency: it is possible to model only the components which have abnormal activity, thereby reducing the size of the model and thus realizing time and space savings.

Discussion Expert systems have been applied extensively within the telecommunications industry, but not without problems. Early (or first generation) expert systems required a knowledge engineer to acquire knowledge from the domain experts and encode this knowledge in a rule-based expert system. These rules were very "ad-hoc" and as the number of rules increased, the expert system became more difficult to understand and modify. The procedure for constructing a rule-based expert system is (a) to define a description language that represents the problem domain, (b) to extract expertise from multiple domain experts and/or trouble-shooting documents, and (c) to represent the expertise in the rule-based reasoning format. The procedure can require several iterations of an interview/implement/test cycle in order to achieve a correct system (Lewis, 1993). If the knowledge does not change very often, little maintenance is necessary. However if the diagnostic expert system is used to solve faults in un-

predictable or rapidly changing domains, two problems inevitably occur (Lewis, 1993). The first one is the *system brittleness*, which means that the system will fail when it is presented with a novel problem. The counterpart of the brittleness problem is the system's lack of ability to adapt existing knowledge to a novel situation or to learn from experience. The second problem is the *knowledge acquisition bottleneck*. It happens when a knowledge engineer tries to devise rules and control procedures that will cover unforeseen situations. When this happens, the system typically becomes unwieldy, unpredictable, and unmaintainable. With rapidly changing domains, the system can become obsolete quickly. The alternatives at this stage are to limit the coverage of the rule-based system or to search for other approaches.

4.2 Model-based Expert systems

Second generation expert systems attempted to solve the limitations of rule-based systems by using stronger methods, such as model-based reasoning. Model-based Systems (MBS) are knowledge-based systems which reason about a system from an explicit representation of its structure and functional behaviour. For the telecommunication networks management, the structural representation involves the description of network elements (NEs) and of the network topology (see Figure 4.3 for an example). The representation of functional behavior describes the processes of event propagation and event correlation (Jakobson and Weissman, 1995). As the real plants tend to be complex, so are the models used in this technique (Penido et al., 1999). Model-based Systems (MBS) have been mainly used in industry for the automation of engineering tasks such as simulation, design, monitoring and diagnosis (Isermann, 1997, 2005; Angeli, 2010). However the same principles can be extended into real-time fault management in a telecommunications network, where the network structure (NE types and topology, containment constraints) and behavior (dynamic process of alarm correlation) are modelled (Gardner and Harle, 1996).

The complexity on building a diagnostic system for Network Management resides on the following facts: a regular network can have a variety of types of hardware components and a large number of them; there are different types of software components (protocols, operating systems, services, applications); the equipment and connections may be changed, and yet some network protocols are based on dynamic configuration. The construction of network models to build management tools involves the identification of all necessary knowledge and its organization in such way that the management task can be automatically performed as an activity of exchanging behavioral, structural and control information (Barros and Lemos, 1999). Several researchers proposed the application of model-based reasoning techniques to solve diagnosis problems.

Kehl et al. (Kehl et al., 1992) presented a generic maintenance system (GMS) for the telecommunication networks (e.g. broadband ISDN networks). The knowledge base of this GMS is divided into two parts: a functional and a physical model, with the functional model being he most important part of it (see Figure 4.3). The functional model consists of structural information and behavioural knowledge. It is built out

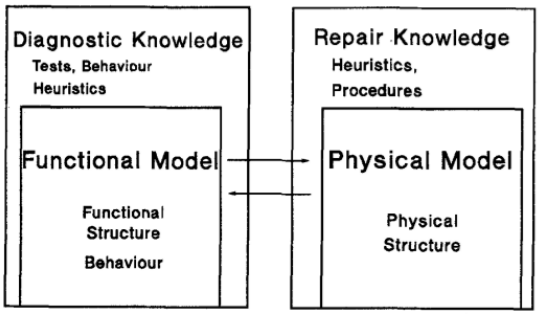


Figure 4.3: Structure of the knowledge sources

of functional entities (FEs) corresponding to specific functionalities of the modelled telecommunication network. There is a mapping between the FEs of the functional model and the elements of the physical model. A functional entity consists out of internal attributes and ports. Such ports connect FEs to each other. The FEs together with these port-to-port connections are building the structural model. FEs have been organized on four different levels of granularity which are connected via a has-subfunctions/is-subfunction-of relation. The reasoning is done on the level which has the finest granularity. Furthermore, a classification hierarchy has been set up over the functional entities: while the upper part of the hierarchy reflects generic telecom classifications, the lower part introduces network-specific classes. The behavioural knowledge is a description of how the network behaves in terms of functional entities and their ports. This generic maintenance system presents the typical characteristics of a model-based system, using an explicit model of the telecommunication network as well as a model of its behavior. It therefore exhibits the advantages of model-based systems including easy maintenance, reconfiguration and extension.

Frohlich et al. (Frohlich et al., 1997) introduced a model-based solution to the problem of alarm correlation in cellular phone networks. They proposed a model called the system description (SD) that consists of a set of axioms characterizing the behavior of system components of certain types while the topology is modeled separately by a set of facts. A set of predicate logic formulas is used to specify the alarm messages as well as their type. And another set of formulas describes the alarm behavior as well as the alarm propagation. The strengths of this alarm correlation system rely on the fact that is is based on: a) a small and maintainable model of the system called the system description (SD) that separates structural or topological knowledge from behavioral knowledge and thus makes changes of the network topology easy; b) a propagation model which allows to correctly diagnose unforeseen errors as well as multiple faults; c) failure probability estimates, which lead to correct diagnoses even on noisy data, where alarm messages have been lost or suppressed.

For building network models, Barros et al. (Barros and Lemos, 1999) specified a Communication Fault Diagnostic System to support the network administrator task. This system is based on the construction of models that represent the network in its

multiple aspects, such as: configuration model, performance model, fault-states causal model, equipment models, and others. In particular, the Configuration Model construction is facilitated by a system, called Network Discovery System, able to collect and gather information about the configuration levels. Since the network configuration is the result of human activity, errors can be embedded in the discovered configuration model. For that reason they have also developed another system called the Configuration Diagnosis System, which can detect a set of configuration errors during the acquisition and construction of the network models.

Dupuy et al. (Dupuy et al., 1991) described a generic network management system (Netmate) to address the management of large, heterogeneous and complex networks. They proposed a model for network management information which emphasizes the definition of generic network objects and relationships. Netmate Structure for Management Information (SMI) defines four object classes and five relationships: 1) *is-in-layer* represents the fact that a node, a link, or a group may belong to a single layer; 2) *is-connected-to* represents the notion that a node (link) may connect to more than one link (node) in the same layer; 3) *is-member-of* represents the collection of groups, nodes and links into a group; 4) *is-part-of* represents the collection of (sub)nodes into a node, or of (sub)links into a link, or of (sub)layers into a layer; 5) *is-implemented-in-terms-of* represents the notion that elements in one layer use the services of elements in other layers, and therefore, are functionally dependent on the well-being of elements in the other layers. These object classes and relationships are able to support various management operations in a number of management scenarios. This object-oriented model is extensible and well suited to accommodate current as well as future heterogeneous networks and protocols.

Miyazawa et al. (Miyazawa and Nishimura, 2011) addressed the issue of the increased time required to identify the root cause of a failure resulting from the increased number and type of alarms caused by network or service failures. They proposed a root cause analysis (RCA) mechanism which classifies types of alarms based on a hierarchical alarm identification type of failure (resource, performance or service failure), but also execute a root cause analysis based on alarm types.

Discussion The model-based approach is easy to deploy and modify and is appropriate for a large-scale network if the network resource information is available (Miyazawa and Nishimura, 2011). Model based systems have the potential to solve novel problems and their performance tends to degrade gracefully when confronted with problems outside their expertise. They also lend themselves well to providing explanations for their decisions and conclusions since each stage in an analysis can be followed and understood. MBSs can be constructed in a modular fashion with different aspects of a physical system being modelled separately, if required, and hence they cater well for expandable, upgradable systems. Also, a model of a system may be used for purposes other than alarm correlation and take different 'views' of the contained knowledge according to the needs of the user or task. However, the application of a model based approach to many systems is often hindered by problem solving complexity. This can be solved, in part, by using more efficient problem-solving algorithms and adopting

the correct system model or sub-models for the the tasks in hand. Selecting the correct level of abstraction for the model is important and the relevant functional, causal, compositional, and structural semantics of the working of the device should be captured (Gardner and Harle, 1996). While these expert systems seem preferable to first generation systems, they have not seen the same level of commercial success. In the field of telecommunications, this is because it is often too difficult to specify a behavioural or functional model at a sufficiently high level to make the model practical and yet have it to be useful (Weiss et al., 1998a).

4.3 Case-based Reasoning systems

Some of the limitations in 'first-generation' expert system technology are addresses in 'second-generation' and 'third-generation' expert programs. The second-generation expert systems use model-based reasoning. This use for network management suffers however, for the difficulty in modeling a complete network, with its interactions in an application (Goyal, 1991). Third-generation expert systems use case-based reasoning and have the capacity to learn naturally with the experience and to avoid the excessive maintenance (Lewis, 1993).

The main idea of case-based reasoning is to recover, adapt, and execute past episodes of problems solution in the evaluation of present problems (Penido et al., 1999). Past episodes are represented in the form of cases in a case library. The experience acquired with the solution proposed is stored in the case library for future references.

Cases contain registers with the most relevant characteristics of past episodes and are stored, retrieved, adapted, and utilized in the solution of new problems. The experience obtained from the solution to these new problems constitutes new cases, which are added to the database for future use.

An important feature of case-based reasoning is its association with learning. When a problem is successfully solved, the parts of the solution which are likely to be useful in the future are stored. When an attempt to solve a problem fails, then the reason for the failure is identified and 'remembered' in order to avoid a recurrence of such a mistake (Gardner and Harle, 1996).

A case generally consists of information about the situation, the solution, the results of using that solution and some attributes that may be used in the searching for similar attributes of other cases. A case-based reasoning system may be described by a cyclic system comprising of four processes (Figure 4.4): *retrieve* the closest matching cases in the past, *reuse* the information in these cases to suggest a solution to the new problem, *revise* the suggested solution in light of testing and *retain* the parts of this experience which may be useful in the future.

Since the development of case-based reasoning systems, several challenges have stimulated the researchers' creativity: how to represent the cases; how to index them to allow their retrieval when necessary; how to adapt an old case to a new situation to generate an original solution; how to test a proposed solution and identify it as either a success or a failure; and how to explain and repair the fault of a suggested solution

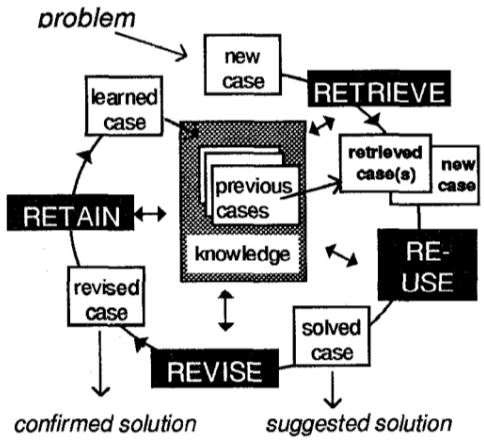


Figure 4.4: Case-Based Reasoning Cycle

to originate a new proposal.

The problem of case adaptation is studied in (Lewis and Dreo, 1993). It described a technique named parametrized adaptation, which is based on the existence in a trouble ticket, of a certain relationship among the variables that describe a problem and the variables that specify the corresponding solution. A CBR system takes into account the parameters of this relationship in the proposition of a solution for the case under analysis. To represent the parameters, the use of linguistic variables (i.e., the ones that assume linguistic values, instead of numeric values) and the provision of functions is proposed, so that the parameters' numeric values are translated into grades of membership in a fuzzy set.

Lewis (Lewis, 1993) reviewed case-based reasoning (CBR) techniques for retrieving, adapting, and embedding knowledge in a case library and described CRITER, a CBR trouble ticketing system for managing an resolving, network faults. The problem of fault management in large and heterogeneous networks is expensive and hard to solve. According to (Lewis, 1993), the solution to this problem includes a) a fault detection system, b) a trouble ticketing management system, c) a representation of fault resolution expertise.

ACS (Penido et al., 1999) is much like CRITER as it has case-based reasoning and it is integrated to the fault management system. However, ACS's approach is to add a case-based component to an alarm collection agent, what makes a decision making in a much earlier stage of fault management. The structure of the database is designed in order to decrease the time it takes to search the cases library by grouping the cases with respect to the kind of the NE and to the category of the problem. ACS's approach tries to reduce the number of alarms the operators can see by correcting faults before sending them to the operators. It is has proved to be greatly helpful for the organization, as it immediately decreased the mean time to repair the faults.

Weiner et al. (Weiner et al., 1995) proposed FIXIT a system for encoding fault man-

agement experience and making it available to operators confronting similar anomalous situations. Its architecture relies on two levels of abstraction to organize fault management information, and on the concept of symptomatic search for retrieving cases. Nevertheless, it presents the disadvantage of typical case-based reasoning systems: it is domain-specific. The semantics of a specific set of fault management experiences and whole-part decompositions do not generalize, except in concept, outside the application for which they were developed. Besides, it does not correct faults.

Melchiors et al. (Melchiors and Tarouco, 1999) proposed DUMBO a system incorporating case-based reasoning paradigm into traditional Trouble Ticket Systems (TTS), that already accumulate knowledge derived from previous problems, to help managers in the diagnosis of a new similar situation. The way DUMBO requests actions and presents solutions is less structured than CRITER. Besides, DUMBO's adaptation is based on the mechanism of context refinement, while CRITER acquires adaptation strategies for the exact solutions. The advantages of DUMBO are its flexibility. Problem types are defined in order to include most of the situations that can be found in the domain. Maintenance and learning are simplified.

Discussion Case-based reasoning (CBR) is an alternative approach to problem-solving that offers potential solutions to the problem of brittleness and knowledge acquisition bottleneck. The goals of CBR systems are (i) to learn from experience, (ii) to offer solutions to novel problems based on past experience, and (iii) to avoid extensive maintenance. The basic idea of CBR is to recall, adapt, and execute episodes of former problem-solving in attempt to deal with a current problem (Lewis, 1993). The technology of case-based systems directly addresses problems found in rule-based systems: First is knowledge acquisition. The unit of knowledge is the case, not the rule. It is easier to articulate, examine, and evaluate cases than rules. Second is performance experience. A case-based system can remember its own performance and modify its behaviour to avoid repeating prior mistakes. Third are adaptive solutions. By reasoning from analogy with past cases, a case-based system should be able to construct solutions to novel problems (Slade, 1991). However, case based reasoning methods suffer from the problem that they must be closely tailored to the domain of application—there are no universal CBR methods. Also, despite the emphasis on the learning process, experience plays a minimal role in refining the method of reasoning; instead they rely on static pre-defined procedures. The time efficiency of the CBR process may also pose problems when it coma to real-time alarm correlation situations. The processing required is potentially complex and time consuming but once up and running, a fast and effective system may evolve which is resilient to changes in network size and configuration (Gardner and Harle, 1996).

4.4 Knowledge discovery and data mining

For many years the telecommunication industry has relied on intelligent solutions to help manage telecommunication networks. Building such applications involved acquir-

ing valuable telecommunication knowledge from human experts and then applying this knowledge typically by embedding it in an expert system. This knowledge acquisition process is so time-consuming that it is referred to as the "knowledge acquisition bottleneck". Data mining techniques can be applied to industrial applications to break this bottleneck by replacing the manual knowledge acquisition process with automated knowledge discovery. Telecommunication networks which routinely generate tremendous amounts of data, are ideal candidates for data mining (Weiss et al., 1998a).

Knowledge discovery is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data (Fayyad et al., 1996). Here, *data* are a set of facts (for example, cases in a database), and *pattern* is an expression in some language describing a subset of the data or a model applicable to the subset. Hence, here, extracting a pattern also designates fitting a model to data; finding structure from data; or in general, making any high-level description of a set of data. The term *process* implies that knowledge discovery comprises many steps, which involves:

- data preparation (selecting, cleaning and preprocessing the data e.g., filling in missing values and transforming it so that it is suitable for data mining);
- data mining (searching for patterns of interest in a particular representational form or a set of such representations, including classification rules or trees, regression, and clustering);
- interpretation and evaluation (interpreting and evaluating the patterns produced by data mining).

The term *nontrivial* means that some search or inference is involved; that is, it is not a straightforward computation of predefined quantities like computing the average value of a set of numbers. The step in the knowledge discovery process which typically requires the most work for telecommunication applications is the transformation step, which involves identifying useful features to represent the data. This step is complicated by the fact that telecommunication networks produce sequences of alarms, where it is not the individual alarms which are of importance but the behavior over time of the network (Weiss et al., 1998a).

Since most data mining methods do not directly operate on temporal sequences, these sequences must be transformed so that these methods can be used. The Scout (Sasisekharan et al., 1993, 1996) application and the forecasting application (Weiss et al., 1998a) both take this approach. An alternative approach is to develop a data mining method which can reason about temporal relationships. The TASA (Hätönen et al., 1996) application follows this alternative approach.

In our brief overview of data mining tasks we use Figure 4.5 as an example. This figure shows a simple two-dimensional artificial data set consisting of 23 cases. Each point on the graph represents a person who has been given a loan by a particular bank at some time in the past. The horizontal axis represents the income of the person; the vertical axis represents the total personal debt of the person (mortgage, car payments, and so on). The data have been classified in two classes: (1) the x's represent persons

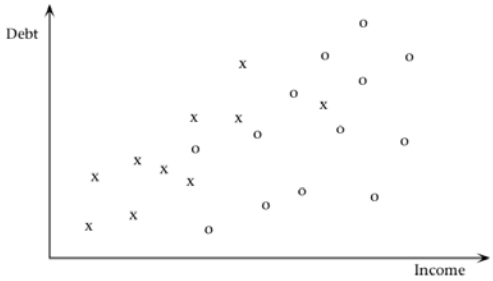


Figure 4.5: A simple data set with two classes used for illustrative purposes (taken from (Fayyad et al., 1996)).

who have defaulted on their loans and (2) the o's represent persons whose loans are in good status with the bank. Thus, this simple artificial data set could represent a historical data set that can contain useful knowledge from the point of view of the bank making the loans. Note that in knowledge discovery applications there are typically many more dimensions (as many as several hundreds) and many more data points (many thousands or even millions).

A typical data mining application from the telecommunications industry is to predict the failure of a network component based on past alarm history. Data mining can be used to solve many tasks, including the following:

- *Classification*: learning a function that maps (classifies) a data item into one or several predefined classes. Figure 4.6 shows a simple partitioning of the loan data into two class regions. The shaped region denotes class *no loan*. The bank might want to use the classification regions to automatically decide whether future loan applicants will be given a loan or not.

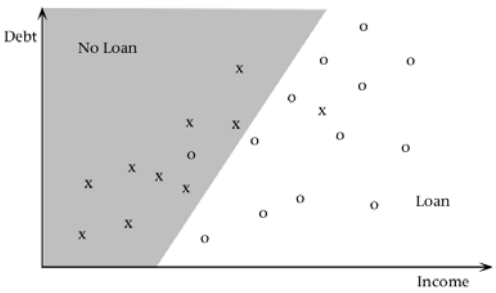


Figure 4.6: A simple linear classification boundary for the loan data set (taken from (Fayyad et al., 1996)).

- *Regression*: learning a function that maps a data item to a real-valued prediction variable. Figure 4.7 shows the result of simple linear regression where total debt

is fitted as a linear function of income: The fit is poor because only a weak correlation exists between the two variables.

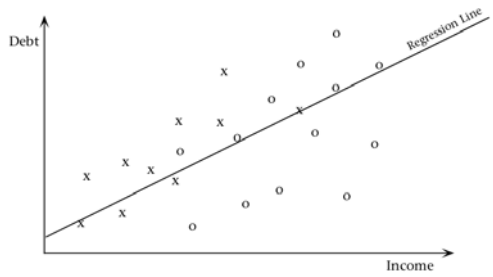


Figure 4.7: A Simple Linear Regression for the Loan Data Set (taken from (Fayyad et al., 1996)).

- *Clustering*: identifying a finite set of categories or clusters to describe the data. Figure 4.8 shows a possible clustering of the loan data set into three clusters; note that the clusters overlap, allowing data points to belong to more than one cluster. The original class labels (denoted by x's and o's in the previous figures) have been replaced by a + to indicate that the class membership is no longer assumed known.

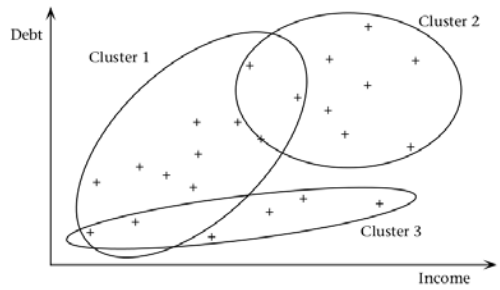


Figure 4.8: A Simple Clustering of the Loan Data Set into Three Clusters (taken from (Fayyad et al., 1996)). Note that original labels are replaced by a +.

- *Dependency modeling*: finding a model that describes significant dependencies between variables. Dependency models exist at two levels: (1) the structural level of the model specifies (often in graphic form) which variables are locally dependent on each other and (2) the quantitative level of the model specifies the strengths of the dependencies using some numeric scale. For example, probabilistic dependency networks use conditional independence to specify the structural aspect of the model and probabilities or correlations to specify the strengths of the dependencies.

- *sequential and temporal pattern detection* deals with the problem of mining patterns from temporal data, which can be either symbolic sequences or numerical

time series. It has the capability to look for interesting correlations or rules in large sets of temporal data, which might be overlooked when the temporal component is ignored or treated as a simple numeric, attribute.

The above tasks can be associated with real telecommunication problems. Applications such as Scout (Sasisekharan et al., 1996), the forecasting application (Weiss et al., 1998a), TASA (Hätönen et al., 1996), and Trouble locator (Chen et al., 1996) can be considered classification tasks (e.g. is a network element faulty or not). While the trouble locator builds dependency models, TASA and to a lesser degree the forecasting application are both example of temporal pattern detection.

AT&T's Scout system proactively identifies recurring transient faults (Sasisekharan et al., 1996). It identifies patterns of chronic problems directly from the data by examining the network behavior over periods of days and weeks. To do this, features which summarize time-varying historical data are extracted from the data so that standard machine learning algorithms can be used (i.e., algorithms which are not capable of explicit temporal reasoning). This featurization is accomplished by using two fixed consecutive time windows, W_1 and W_2. The objective is to use the measurements from W_1 to predict problems in W_2. One way of summarizing these measurements is to count the number of times each feature occurs within the window. Scout then used Swap-1 (Weiss and Indurkhya, 1993) to learn rules that predict recurring transient faults.

The main objective of the forecasting application (Weiss et al., 1998a) is to identify patterns of messages that identifies the pending arrival of crucial events namely those corresponding to catastrophic failure of switching equipment. The (initial) time-series data are expressed in a standard case-based representation, where each case includes a set of feature variables and a single class variable. This transformation is accomplished by using two fixed consecutive time windows, a monitor window M and a predictive window P. The objective is to observe the count of each message type in M then look forward some fixed unit of time and observe in P whether the crucial event occurred. Each case is represented by a monitor window, M, and a prediction window, P. Once time-series data are transformed into cases, standard machine learning classification methods such as neural nets or decision trees, can then be applied to the transformed data. Predictive performance is maximized by varying a sampling period window and a prediction period window during the data transformation.

The Telecommunication Network Alarm Sequence Analyzer (TASA) (Hätönen et al., 1996) is a system for locating regularities in the alarm sequences in order to filter redundant alarms, locate problems in the network and predict future faults. TASA operates in two phases. In the first phase, specialized algorithms are used to find rules that describe frequently occurring alarm episodes from the sequential alarm data (Mannila et al., 1995). An episode describes a set of alarm sequences over a given time period and this set can include alarm sequences in which the specific order of alarms does not matter. In the second phase, collections of episodes are interactively manipulated by the user so that interesting episodes from the original set can be found. TASA supports this process by providing operations to prune uninteresting episodes, order the set of

episodes and group similar episodes.

Pacific Bell has an intelligent system which determines the location of troubles in a local telephone cable network (Chen et al., 1996). This system uses data generated by a nightly automated test to help narrow down potential cables or network equipment which may be faulty; however, the test results are not sufficient to determine the exact cause. The Trouble Locator uses a Bayesian network and Bayesian inference (Pearl, 1988) to solve this problem. The system begins by generating a local plant topology graph and then from this generates a Bayesian network, where each node in the network contains state information (belief of failure) of a plant component. This network also takes into account historical information about the components and the data from the overnight test. The belief of failure is then propagated throughout the network until equilibrium is reached, at which point a ranked list of faulty components can be generated. This system is used by preventative maintenance analysts as a decision support system.

Data mining methods for solving the various data mining tasks vary in several ways, including: the time they require for learning, their tolerance of noise, the expected format of the data and the concepts they are capable of expressing. Rule induction (used in Scout (Sasisekharan et al., 1996)) and Bayesian networks (used in (Chen et al., 1996)) are two data mining methods used extensively within the telecommunications industry. Other data mining methods, such as neural networks can also be used to solve data mining tasks. Indeed, non linear regression and classification methods are a family of techniques for prediction that fit linear and non-linear combinations of basis functions to combinations of the input variables, and examples of such methods include feed forward neural networks. Figure 4.9 illustrates the type of nonlinear decision boundary that a neural network might find for the loan data set.

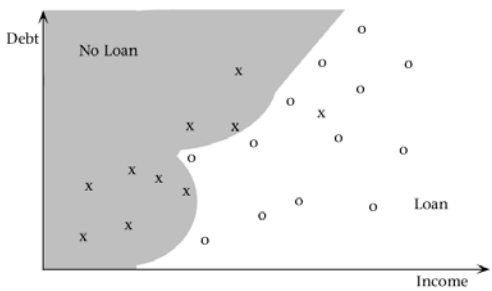


Figure 4.9: An Example of Classification Boundaries Learned by a Nonlinear Classifier (Such as a Neural Network) for the Loan Data Set (taken from (Fayyad et al., 1996)).

An Artificial Neural Network (ANN) is a system comprising a number of mutually connected elements (neurons) having a simple input-output. Conceptually, each neuron may be considered as an autonomous processing unit, provided with local memory and with unidirectional channels for the communication with other neurons. The functioning of an input channel in an ANN is inspired by the operation of a dendrite in biological neurons. In an analog way, an output channel has an axon as its model. A

neuron has only one axon, but it may have an arbitrary number of dendrites. The output "signal" of a neuron may be utilized as the input for an arbitrary number of neurons. In its simplest form, the processing carried out in a neuron consists of affecting the weighted sum of the signals present in their inputs and of generating an output signal if the result of the sum surpasses a certain threshold.

Feedforward neural networks have already been proven effective in medical diagnosis, target tracking from multiple sensors, and image/data compression. It is therefore plausible that NNs would be effective for the similar problem of alarm correlation, found in fault diagnosis. In a feedforward neural net, shown in Figure 4.10, the neurons are arranged into layers, with the outputs of each layer feeding into the next layer. This model has a single input layer, a single output layer, and zero, one, or more hidden layers. As the name suggests, all connections are in the forward direction where there is no feedback. Feedforward networks are useful because of their ability to approximate any function, given enough neurons, and their ability to learn (generalize) from samples of input-output pairs. Learning is accomplished by adjusting the connection weights in response to input-output pairs, and training can be done either off-line, or on-line during actual use. Depending on how the training is done, these NNs can be characterized as being trained by supervised methods or by unsupervised methods.

Supervised NNs training data consists of correct input vector/output vector pairs as examples, used to adjust the neural net connection weights. An input vector is applied to the NN, the output vector obtained from the NN is compared with the correct output vector, and the connection weights are changed to minimize the difference. A well trained neural net can successfully generalize what it has learned from the training set (i.e., given an input vector not in the training set, it produces the correct output vector most of the time).

In unsupervised training there is no training data based on known input/output pairs. The NN discovers patterns, regularities, correlations, or categories in the input data and accounts for them in the output. For example, an unsupervised neural net where the variance of the output is minimized could serve as a categorizer which clusters inputs into various groups. Unsupervised training is typically faster than supervised training and provides the opportunity to present patterns to operations personnel who can identify new output relations. For these reasons unsupervised training is used even in situations where supervised training is possible. However, for the domain of alarm correlation, input/output pairs can be easily produced, making supervised trained NNs a plausible choice for alarm correlation.

A different neural net approach for alarm correlation (Patton et al., 1994) uses the ability of neural networks to predict future behavior of general nonlinear dynamic systems. In this approach, a neural network predicts normal system behavior based on past observations and the current state of the system. A residual signal is generated based on a comparison between the actual and predicted behavior, and a second neural network is trained to detect and classify the alarms based on characteristics of the residual signal. This method can be used to identify basic categories for the alarms.

An additional approach is to cast the pattern recognition problem into an optimization problem, making Hopfield NNs an appropriate tool for alarm correlation. This type

of NN operates by using gradient methods to find a local minimum of a quadratic energy function that represents an optimization problem, and whose coefficients depend on the network's interconnection strengths. Methods such as mean-field annealing, repeated trials with random initial states, and tabu learning (Beyer and Ogier, 1991) can be used to find a local minimum that is nearly optimal. For example, in alarm correlation, the optimization problem is to identify the hypothesis that best explains the observed data. (Goel et al., 1998) propose a neural network model based on Hopfield nets for solving a special case of this problem. In this case, the neural net would be designed so that states corresponding to the most likely hypotheses have the lowest energy.

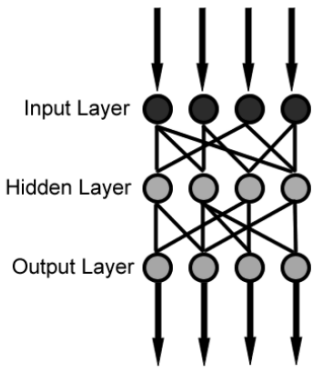


Figure 4.10: Model of a feedforward neural network.

Discussion The difficulties of using knowledge discovery and data mining include considerations such as the availability of sufficient data (cases). In general, the more fields there are and the more complex the patterns being sought, the more data are needed. Another consideration is the relevance of attributes. It is important to have data attributes that are relevant to the discovery task; no amount of data will allow prediction based on attributes that do not capture the required information. Furthermore, low noise levels (few data errors) are another consideration. High amounts of noise make it hard to identify patterns unless a large number of cases can mitigate random noise and help clarify the aggregate patterns. Finally, and perhaps one of the most important considerations, is prior knowledge. It is useful to know something about the domain—what are the important fields, what are the likely relationships, what is the user utility function, what patterns are already known, and so on (Fayyad et al., 1996).

The following properties of multilayer feedforward neural networks make them a powerful tool for alarm correlation.

- NNs can recognize conditions similar to previous conditions for which the solution is known (i.e., pattern matching).
- They can approximate any function, given enough neurons, including boolean functions and classifiers. This gives NNs great flexibility in being able to be

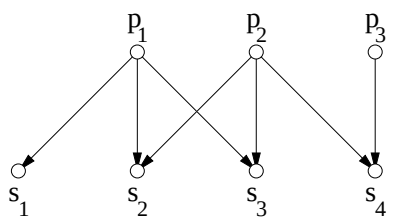
trained for different alarm patterns.

- They can generalize well and learn an approximation of a given function, without requiring a deep understanding of the knowledge domain.
- They provide a fast and efficient method for analyzing incoming alarms.
- They can handle incomplete, ambiguous, and imperfect data.

They have been used for the purpose of fault localization in (Gardner and Harle, 1997, 1998; Wietgrefe, 2002). Nevertheless neural networks have the disadvantage that they require intensive training before being able to associate an output pattern with a given input pattern (Gardner and Harle, 1996; Wu et al., 1998). Although this learning process must occur, it is not always convenient in a telecommunications environment where all the alarm signatures of fault occurrences may not be known or are not readily available.

4.5 Fault-symptom graphs and the codebook approach

The codebook approach to event correlation consists in associating the n different problems $p_1, ..., p_n$ one wishes to recognize to patterns of k observable symptoms $s_1, ..., s_k$. Each problem p_i is thus characterized by an alarm vector, checking the presence/absence of all the selected symptoms. If the symptoms are numerous enough and correctly chosen, they allow not only to discriminate between problems $p_1, ..., p_n$, but also to resist to a number of observation errors, either false alarms (spurious symptoms) or misdetections (missing symptoms or lost alarms). This technique has been popular in a number of problems for its robustness properties (Reali and Monacelli, 2009; Tang et al., 2008; Frohlich et al., 1997); it generally proceeds by analysing the alarms within a sliding time window. There exists a probabilistic version of this approach, and several techniques have been proposed to automatically build a model relating problems to symptoms and to optimally select the relevant sets of symptoms. Some contributions also explain how to adapt the codebook to network changes (Yemini et al., 1996).



	p_1	p_2	p_3
s_1	1	0	0
s_2	1	1	0
s_3	1	1	0
s_4	0	1	1

Figure 4.11: A problem-symptom graph and its associated codebook.

The principle. Consider problems $p_1, ..., p_n$ to be diagnosed, and the observable symptoms $s_1, ..., s_k$. A codebook is a binary matrix $C \in \{0, 1\}^{k \times n}$, where $C_{i,j} = 1$ iff

problem p_j can produce symptom s_i. Each column $C_{.,j}$ is thus the signature (or the codeword) of problem p_j, and each row $C_{i,.}$ represents the sensitivity of symptom s_i to the different problems. Equivalently, one can view this matrix as a simple bipartite causality graph relating problems to symptoms (see Figure 4.11).

A minimum of $k = \lceil log_2 n \rceil$ symptoms (bits) is sufficient to distinguish the n different problems, but k is generally larger because: a) one does not fully master the sensitivity of symptoms, and some selection should be made (see below), b) and having more symptoms than necessary ensures some robustness to errors in the values of the symptoms. There is actually a straightforward analogy with error correcting codes: the codeword $C_{.,i}$ of problem p_i should be as far apart as possible from that of problem p_j, $j \neq i$, where the distance can be chosen as the Hamming distance d_H, which counts the number of different bits. With minimal distance $d_{min} = \min_{i \neq j} d_H(C_{.,i}, C_{.,j})$ between codewords, one can correct at most $r = (d_{min} - 1)/2$ observation errors. Let $(o_1, ..., o_k) \in \{0,1\}^k$ be an observation vector ($o_i = 1$ iff symptom s_i is present), then any observation vector within the ball of radius r around codeword $C_{.,i}$ will be "decoded" as problem p_i. The analogy stops here, however: by construction, the codebook does not correspond to a linear code, so all usual (fast) decoding strategies are unavailable. One is thus bound to decoding an element of $\{0,1\}^k$ by table lookup, minimizing the (Hamming) distance to the different codewords.

This approach is most often presented as a fast and robust single problem identification method. It extends to multiple problems provided one can reasonably assume that symptoms accumulate (i.e. one can add up the codewords bit by bit, with the rule 1+1=1). The problem then becomes that of finding the minimal number of problems that best explains/fits the observed vector of alarms.

Codebook construction. This point covers three aspects:

1. the derivation of a causality graph connecting problems to symptoms,
2. its adaptation to network changes, and
3. the selection of the relevant symptoms in order to ensure robustness.

A starting point to derive the problem-symptom relations is a causality graph representing the propagation of failures from the root causes to the observable alarms (see also the section dedicated to causality graphs). This causality graph contains much more information/knowledge than necessary to the codebook approach, in particular it encodes problem dependencies, the presence of intermediary consequences, and numerous possible symptoms. The causal graph can thus be considered as a relevant model for inference (see causal graph inference, and the Bayesian networks section), that would moreover provide an explanation to the collected observations. Here, one rather looks for a fast and robust technique to quickly diagnose a problem. It thus proceeds by extracting characteristic alarm patterns from this graph. The first step consists in removing all intermediary nodes, i.e. those not identified as a problem or as an alarm/symptom, by taking the transitivity closure of the causality relation (assumed

acyclic). The causal relations between problems are dropped: problems are assumed independent. And redundant symptoms are also discarded (i.e. symptoms that duplicate the behavior of other symtoms). Figure 4.12 illustrates a causal graph that could lead to the problem-symptom graph in Figure 4.11.

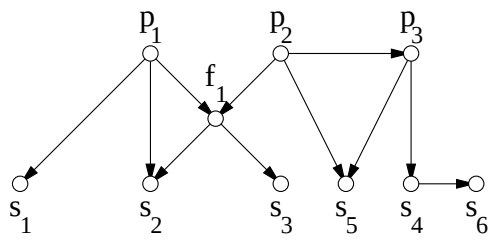


Figure 4.12: A causal graph explaining the propagation of failures from the problems (top) to the symptoms (bottom). This causal graph yields the problem-symptom graph of Figure 4.11 by transitivity closure, after removal or intermediate nodes and duplicated symptoms.

This construction brings the problem back to the derivation of a relevant causality graph for a given network, a richer structure than the problem-symptom graph. The latter can derive from expert knowledge (see the section dedicated to causality graphs). For example, it can result from an in depth analysis of failure scenarios and the attached alarm logs. In other words, it can be identified or learnt from a collection of actual network data. Alternately, some contributions have proposed to derive a causality graph from the very structure of the network, which resembles the first step of model-based approaches. In (Yemini et al., 1996), the authors propose to model several network layers as a collection of connected objects. The ideas presented there are quite appealing, despite a weak formalization. They proceed by identifying the relevant classes of managed objects (*physical objects, logical objects, links, nodes*) and their relationships (*consists of, contains, belongs-to-layer, layer-over/layer-below, provides/consumes, serves/served-by, connected-to/connected-via,* etc.), following the Netmate class hierarchy. A given network is then composed of instances of such objects and of their relations, which gives a first topological structure of the managed object dependencies and relations. Within each class, one then identifies the events that can appear, either as problems that can propagate to connected objects, or as alarms that can be reported to the manager. One also needs to model how objects react to events in their neighbors. For example, the event (problem) *SyncLoss* in an object of class *WAN-Interface* can propagate to the objects of class *link* through the *connected-to* relation, and produce an event *BitErrorRate-above-threshold* in these links. The last step consists in labeling the events in the causal graph either as problems, intermediary events or symptoms, and proceed to the extraction of the problem-symptom graph. The main advantage of this approach is that it can adapt the knowledge used for diagnosis to the evolution of the network topology, one of the few contributions with this much desirable property.

Building an adequate problem-symptom graph is not sufficient to ensure the efficiency and the robustness of the diagnosis engine. A trade-off is necessary between the selection of the most discriminant symtoms (efficiency) and the preservation of a sufficient level of redundancy (robustness). This is generally performed in a two-stage manner. The selection first isolates a subset of symptoms (at least $k = \lceil log_2 n \rceil$) that best discriminate the n problems $p_1, ..., p_n$. This consists in building a coding for these n symbols. The approach is essentially empirical, and relies on the successive selection of symptoms s_i that have a balanced number of 0 and 1 in their sensitivity pattern $C_{i,.}$. One has to guarantee that any added bit (or symptom) in the coding maximally reduces the entropy of the "problem" variable P that takes its values in the set $\{p_1, ..., p_n\}$. For example, a symptom reacting to a single problem is of little interest. Unfortunately, the problem is never stated as a source compression question, which would definitely be useful there, in particular to take problem likelihoods into account. Once a valid minimal coding is available, the issue of robustness is also addressed by heuristics. The idea is to incorporate extra symptoms in order to maximize the minimal (or average) Huffman distance between codewords.

Probabilistic version. Causal graphs have a direct extension as Bayesian networks. However, this is not the clean track that has been followed by the codebook approach. A first difference lies in the definition of problem likelihoods, which is generally ignored. This translates the fact that there is an ambiguity between the identification of a single problem, which would mean assigning a distribution over the set $\{p_0, p_1, ..., p_n\}$ where p_0 represents the "no failure" case, and the identification of a multiple independent problems, which would mean assigning a distribution to a n-uple of binary variables $P_1, ..., P_n$. This second case corresponds to a classical Bayesian network approach.

The probabilistic version of the codebook approach rather focuses on modeling the likelihood of the different symptoms conditionally to the problems (Kliger et al., 1995). When there exists a connection between problem p_j and symptom s_i, the model specifies $\mathbb{P}(s_i = 1|p_j = 1) = d_{i|j}$, i.e. the probability of detection (or misdetection, by complement). False alarms are thus not assumed... If a symptom depends on several problems, the assembling of these specifications is performed by the noisy OR technique. For example, assume s_i depends on p_j, p_k, p_l, one then states $1 - \mathbb{P}(s_j = 1|p_j = 1, p_k = 1, p_l = 0) = (1 - d_{i|j})(1 - d_{i|k})$, and similarly for other values of the triple (p_j, p_k, p_l). This models the fact that only misdetections can occur, and that these misdetections are independent when several problems are present. Such likelihoods are then used to compute distances from codewords to problems. In addition, one can incorporate extra probabilities for erroneous detections of symptoms, and to account for false alarms. The decoding is then performed by a minimal distance criterion, i.e. on the basis of a maximum *a priori* principle.

As one can notice, this randomization of the model is *ad hoc* and rather questionable on many points. The derivation of a probabilistic problem-sympton graph from a probabilistic causal graph also suffers from several weaknesses: it ignores the distribution of problems and their correlations, it also ignores the correlation between symptoms introduced by intermediary malfunctions such as f_1 in Figure 4.12: one can

not anymore assume that symptoms (as s_2 and s_3) are conditionally independent given the problems, as it is erroneously presented in (Kliger et al., 1995). Bayesian networks provide a more appropriate framework to encode such causality models and to perform a more rigorous estimation of the problems causing the observed symptoms.

Discussion. The codebook approach is a rather fast and flexible pattern recognition technique that identifies problems from the characteristic patterns of symptoms that they produce. Its strong points are the simplicity and the adjustable robustness. Some contributions have proposed a methodology to tailor the codebook to a specific network instance, and to adjust it as the network evolves.

However, this technique suffers from numerous drawbacks. The most obvious ones are that it focuses on the identification of a single problem, and that it provides no explanation to what happened in the network. This is a little bothering since the very construction of the problem-symptom graph uses an elaborate object, namely a causal graph, that partly has this explanation capability and a more powerful description power. On the formal side, both the construction and optimization of the codebook and the decoding rely on heuristics. The probabilistic version of the model is too weak, just like the handling of time. It seems that model-based approaches using directly causal graphs, or Bayesian networks for their probabilistic version, are much more elegant. For the same modeling effort, they provide explanations about failure propagations, deal with multiple failures, capture false or missed alarms, and rely on some inference algorithms.

4.6 Dependency/causality graphs

Fault-symptom graphs can be considered as the basic level of model-based approaches to event correlation. By contrast, causality graphs, and their close siblings dependency graphs, make a step further towards a more accurate modeling of network structures and behaviors. And as mentioned above, their construction is a natural prerequisite to the derivation of an accurate fault-symptom graph. So it seems natural to directly work with causal graphs, and take advantage of their sparsity, which mechanically vanishes once intermediary faults are removed.

The principle of causal graphs is to represent not only the initial failures (root causes) and the symptoms that they generate, but also to model the complex chains of intermediate faults and failures that relate them (Kätker and Paterok, 1997). This knowledge derives from several types of information: expert knowledge about the mechanisms of fault propagation and alarm production, but also the dependency relations between softwares and equipment, which are inherent to the very design of such complex systems as networks. By contrast with codebook approaches, the inference consists in guessing the state of *all* relevant nodes in a causality graph, once some observation nodes (alarms) are positioned to true or false. This is again a static approach to event management, in the sense that it operates on a snapshot of the network state, as opposed to dynamic approaches that track the evolution of this state.

Causal graphs and their variations. Several formalisms of causal graphs have been proposed in the literature. We focus first on the work of (Lu et al., 2010, 2011) which is quite exemplary. In this setting, a causal graph is first of all an oriented graph $G = (V, E)$, and specifically a DAG (Directed Acyclic Graph). Vertices (or nodes) in V represent events, and arrows in $E \subseteq V \times V$ stand for causality relations. A node is minimal iff it has no predecessor for E, and maximal iff it has no successor.

Vertices can be of several types: minimal nodes (for the partial defined by the edges) are generally called primary causes and represent the failures one wishes to identify, nodes both with predecessors and with successors are simply intermediate faults, and maximal nodes represent observable events. The latter can be further divided into immediately observable nodes (i.e. alarms), test nodes (which provide an observation only if they are triggered), or display nodes. These display nodes are purely indicative and play no part in the semantics of the graph: they are there to express a diagnosis hypothesis, and/or to suggest a repair action. The latter could then be triggered automatically if such a node assumes value *true*, but its effect on the causal graph is unspecified. A *configuration* of the graph is a function $c : V \to \{false, suspect, true, unknown\}$ that associates a status to each node: *true* (resp. *suspect*, resp. $false$) represents the presence (resp. suspected presence, resp. absence) of a root cause or of an intermediate fault. The value *suspect* is forbidden for terminal nodes (i.e. alarms, tests and displays), since they represent the presence or absence of an alarm, or the fact that this observation is simply not known.

The edges of E represent causality relations between events, and can assume different modalities, as in modal automata. For example, one can partition E into $E = E_{must} \uplus E_{may}$. Edge $(a, b) \in E_{must}$ means that the failure of a causes the failure of b, whereas $(a, b) \in E_{may}$ means that the failure a can be the cause of the failure of b, but this is not mandatory: the propagation can stop. In other words, configurations where $a = true$ and $b \neq true$ are not allowed in the first case. A configuration c is said to be consistent (see Figure 4.13) iff edge modalities are satisfied, and for all non minimal node a such that $c(a) = true$, at least one of its predecessors assumes value *true* (i.e. there is a cause to the failure at a).

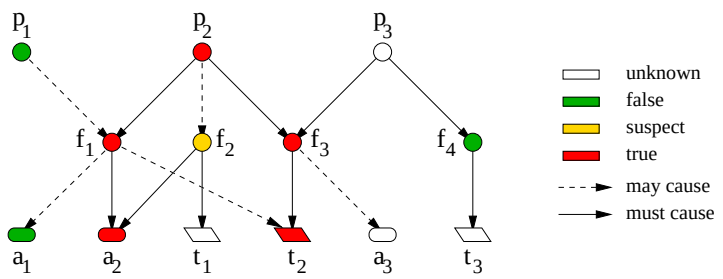


Figure 4.13: A causal graph and a configuration over it. Edges describe two propagation modalities: "must cause" (solid arrows) and "may cause" (dashed arows). Test nodes are represented as parallelograms, alarm nodes as ovals.

By definition, observable nodes are always terminal. Nothing prevents having them

at any place in the graph; this simply translates the fact that the consequences of an observable node (i.e. alarms resulting from other alarms) can only trigger redundant alarms. Which implicitly assumes that (intermediate) alarms cannot be lost, an assumption that is not made in fault-symptom graphs, nor in Bayesian networks, the probabilistic version of causal graphs.

The literature proposes numerous variations around causal graphs. One can actually play with node types, with propagation modalities, and with the possible values that each node can take. Some contributions have considered causal graphs as a representation of logical constraints on the values that nodes can assume, i.e. on valid configurations. Implicitly, in the above setting, the edges going out of a node represent an AND on the consequences of that node, while the incoming edges represent an OR on the possible causes of that node. (Hasan et al., 1999) goes further by allowing any logical constraint relating a node to its offsprings, in the binary case. For example, one can encode the relation $a \Rightarrow b\ OR\ (\ c\ AND\ d\ )$ when b, c, d are the offsprings of a. One could go further in this direction and consider a more general formalism where nodes represent variables (taking values in any domain), and the graph encodes constraints relating these variables. Specifically, a constraint between variables of $W \subseteq V$ makes W a clique in the (non-oriented) graph, that is all pairs of nodes in W are connected. This setting is a degenerated case of Bayesian networks, which allows to recycle their inference algorithms (Dechter, 2003).

Inference procedures. The diagnosis proceeds as follows. For a given time window, and thus for a given snapshot of the network state, one positions the observed alarms to $true$ in the causal graph. Negative symptoms may also be observed (for example the fact that some connectivity or service is still alive) and thus positioned to $false$. All other terminal nodes are positioned to $unknown$. The objective is to compute the set C of all configurations c explaining the observed nodes. One is only interested in the root causes, i.e. in the minimal nodes of configurations c that take value $true$: this is called a *possible diagnosis* for the observed malfunctions. For example, in Figure 4.13, one has $a_2 = true$ and $a_1 = false$ as observations, plus the result of test t_2 which returned $t_2 = true$. A possible diagnosis is the initial failure $p_2 = true$. Among all elements of c, some may contain numerous root causes that generate the same observations. In that case, Occam's razor applies and one selects the configurations with the minimal number of root causes positioned to $true$, and proposes them as the most likely explanations.

The algorithm proceeds recursively, in a bottom-up manner, starting from the observed symptoms and trying to guess what are the possible causes of the observed alarms (there must be at least one, by construction). The value *suspect* is used for that purpose, to keep track of the opened hypotheses about possible causes. The algorithm is thus a simple constraint propagation procedure, which branches when several possibilities appear. Consequently, when the value of a node is determined (either $true$ or $false$), it can not be reconsidered. Observe that the effects of failures are "additive" by construction: the initial failure b can only add to the consequences observed for the initial failure a. This simplifies much the inference: one does not have to explore complex patterns of failures that would make some symptom disappear (inference would

then become NP hard). Instead, the problem "simply" amounts to finding at least one cause to every observed symptom: only missing (positive) symptoms can invalidate a suspected cause, and since minimal explanations are desired, there is no need to further add extra causes with the same effects.

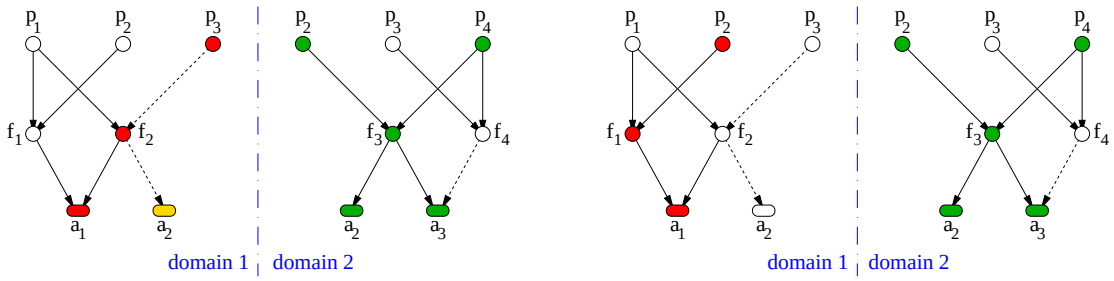


Figure 4.14: Two causal graphs, for resp. domains 1 and 2, sharing some nodes (p_2, p_3 and a_2). Left: two compatible local configurations. Right: two incompatible local confidurations.

The formalism proposed in (Lu et al., 2010, 2011) presents two exciting features. A first one is the possibility to perform tests, and thus to acquire information that will allow one to confirm or discard some diagnosis assumptions. The question of which test would be the most informative remains open: one would actually like to reduce as much as possible the set of current diagnosis assumptions (configurations) that are being explored by the algorithm. This is a first step towards *active diagnosis procedures* that cleverly poll the network to retrieve more information. A second feature is the ability to perform a distributed diagnosis. The idea consists in separating the causal graph into several subgraphs, sharing some nodes. Each subgraph is the causal graph of some component or domain of the network. The diagnosis can be performed independently in each subgraph, and followed by a message passing procedure between neighboring components in order to check the compatibility of the proposed local configurations (see Figure 4.14). This is a specialized version of distributed constraint solving algorithms as they appear in (Dechter, 2003), or equivalently a specialization of the belief propagation (or message passing) algorithms that one can design for Bayesian networks (Pearl, 1988). Interestingly, checking the compatibility of two local configurations can be done at low complexity, by considering only their restriction to the common variables (see (Lu et al., 2011) for details).

Several inference mechanisms have appeared in the literature, more or less in the same spirit. For example (Hasan et al., 1999) translates the logical relations relating each node to its offsprings into a collection of local inference rules that allow to determine the value of a node from that of its offsprings. The originality of this work is to extend the setting to temporal relations. Under the direct translation into a rule language, the inference mechanism is basic and does not allow for branchings and multiple assumptions. The most general inference machinery one could expect from general causal graphs would certainly be the family of SAT solvers, although apparently no publication refers to it.

Interestingly, (Grosclaude, 2008) proposes a slightly different approach, when test nodes are more numerous than alarm nodes. The idea is to *compile* the causal graph into a diagnosis procedure, that selects the most relevant tests to perform and their ordering, in view of quickly identifying the origin of a failure. This results into a decision diagram driven by tests, as the ones that are used by hotline operators. Apparently, the research direction has not been fully formalized nor explored.

Graph construction. The major bottleneck of this approach is clearly the derivation of the causal graph: given this graph, the design of the inference engine is rather easy and is a well paved road. Unfortunately, the difficulties on the modeling side have received much less attention. From the few papers that address them, there are essentially two sources of information that can be used.

1. Dependency graphs, or *modeling how things work*. This consists in selecting the set of network resources that will appear in the model, together with their possible failures and the symptoms they produce (Houck et al., 1995). One then represents the dependencies that relate these resources. In (Gruschke et al., 1998), these resources are split into 3 categories: services (connectivity, access to a data-base, etc.), the physical equipment, under the form of end-systems, and the network devices, that guarantee the physical connectivity between end-systems. The dependency relations between these resources derive from the network topology, the MIBs (Management Information Bases) of the equipment and of the applications, or the inventory systems. An important concern is the definition of the appropriate granularity for the model. Interestingly, this paper advocates the design of a user interface allowing an expert to partially build and refine this model. It also advocates the *construction at runtime* of the appropriate dependency graph, in order to track topological changes in the network. In the same spirit, (Grosclaude, 2008) (in French) makes use of operational models, at the service level, that capture some of the resources (or prerequisites) necessary to a service, for example HiFi VoIP. These dependencies describe both structural requirements as well as temporal requirements. Temporal requirements express for example the correct execution of protocols or procedures that are necessary to launch a service (e.g. establishing a connection, obtaining an IP address, having internet connectivity, etc.). As another source of information, dependency relations can also be derived from the information models that define how the managed objects are structured, connected and related (Choi et al., 1999; Yemini et al., 1996; Houck et al., 1995).

2. Failure graphs, or *modeling why things may not work*. This is the dual approach, that consists in examining failure cases. This information generally comes from experienced situations (as in Cased Based Reasoning) compiled by experts into failure scenarios. It provides *partial* fault propagation examples that can be directly translated into a causal graph (Grosclaude, 2008). This model can be enriched with test scenarios or simple information retrieval that can enrich the observations (for example asking the user the color of a led on his equipment). However, it is by essence incomplete, limited to/by the interpretation of experts

and sometimes erroneous, limited to the experienced failures, and it can also be too much specialized to a given network topology.

Discussion. Compared to fault-symptom graphs, causal graphs offer several interesting advantages: beyond the detection of the modeled failures (root causes), they also explain what happened in the network as consequences of these failures. This is a valuable information for the operator that can then check the validity of these automatic interpretations of alarms, and further check the relevance of the fault model. Moreover, causal graphs are generally a prerequisite to the derivation of fault-symptom graphs, but they have a sparser structure which is beneficial to inference algorithms.

Several formalisms have been proposed for causal graphs, that encode various kinds of logical relations between causes and effects. As they are designed for snapshots on the networks (i.e. for events collected within a time window), they grab both strict causalities, but also precedence, possibly with time constraints. The expressivity level of the different formalisms varies, which results in inference problems of different complexities. But overall, all can be captured under the umbrella of SAT problems. Inference is well understood for this family of problems, and even admits distributed/modular versions. In particular, one can consider systems defined by local (logical) constraints as specific cases of Bayesian networks, for which exact and approximate inference algorithms are available and rather well understood. Interestingly, some contributions have highlighted that the possibility to perform tests was raising new problems, like the optimal selection of these tests, which is a first step towards active diagnosis. Further, one can also compile offline a causal graph into organized test plans that would help resolve some observed malfunctions, and that could be used, for example, as a support for hotline operators.

While algorithmic resources are well mastered for these problems, the more demanding (but necessary) modeling stage has been much less explored. Some contributions have proposed to use and compile the available knowledge that can be derived from topological considerations: managed objects definitions and structures (information models), network resources dependencies (MIBs), protocols and procedures necessary to the establishment of services, etc. Some contributions have suggested that this could open the way to "modelling at runtime", in order to fit the model to the managed network. Two important questions remain open: the selection of the appropriate granularity for the model, and the definition of an interactive modeling framework that would help an operator or an expert check and refine the current version of the model.

4.7 Chronicles: patterns of timed events

Principle. This approach proposes to monitor a continuous flow of timed events $a_1, a_2, ..., a_n, ...$, and to look for known patterns of events within this flow that characterize a specific malfunction (Dousson, 1996; Dousson and Maigat, 2007).

The flow is made of the observable events that reach the network manager, and each event a_n is characterized by some event type $\tau(a_n)$ denoting the nature of the

observed problem (for example "excessive BER on equipment E") and by a date $\delta(a_n)$ at which this alarm was produced. These dates assume the existence of a global clock in the netwok.

The patterns one is looking for in this flow are called *chronicles*. A chronicle is a labeled oriented graph of events $G = (V, E, \tau, \alpha, \beta)$. Vertices V represent events, and τ associates an event type to each of them. Edges $E \subseteq V \times V$ encode temporal dependencies between the events they connect, by means of functions $\alpha, \beta : E \to \mathbb{R}$. To the edge $e = (v, v') \in E$, better denoted $v \to v'$, one associates the time interval $I(e) = [\alpha(e), \beta(e)]$ which means that event v' must appear at least $\alpha(e)$ time units and at most $\beta(e)$ time units after v. Equivalently, if events v and v' appear at dates t and t' respectively, then one must have $\alpha(e) \leq t' - t \leq \beta(e)$. Observe that one could very well reverse the orientation of an edge $e = (v, v')$ without changing this semantics, by taking $I(v' \to v) = -I(v \to v') = [-\beta(e), -\alpha(e)]$. So the oriented graph G need not be acyclic. Observe also that the time intervals $I(e)$ may contain both negative and positive values, therefore a chronicle does not necessarily encode causality relations between events, although this is generally the case in practice.

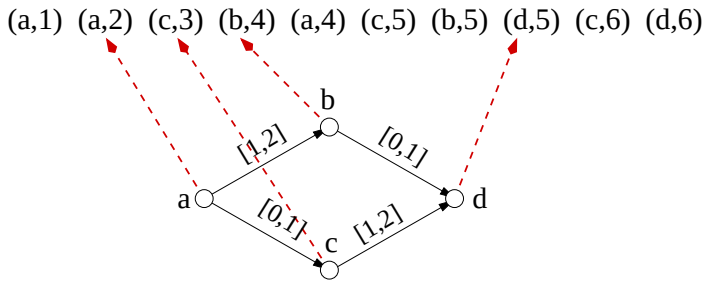


Figure 4.15: An timed event flow (top) and a chronicle (bottom). The arrows indicate a chronicle instance within the event flow.

An *instance* of chronicle $G = (V, E, \tau, \alpha, \beta)$ into the alarm flow $(a_n)_{n>0}$ is a mapping $\phi : V \to \{a_n, n > 0\}$ that preserves event types and such that event dates satisfy the time contraints of the chronicle (Figure 4.15). Specifically, one must have $\forall v \in V,\ \tau(\phi(v)) = \tau(v)$, and $\forall e = (v, v') \in E,\ \delta(\phi(v')) - \delta(\phi(v)) \in I(e)$.

For simplicity here, we consider the basic language of chronicles, that only considers event occurrences. The formalism extends to capture the absence of events within a given time interval, the continuous presence of a phenomenon within some period, or to count the number of events of a certain type within some period.

Chronicle consistency. We have mentioned above that G needs not be acyclic. Nevertheless, the set of time contraints that it defines must be consistent. This verification can be performed by propagating the time constraint of each edge to the rest of the graph, until stability. One starts by completing G with all missing edges $v \to v'$ (provided $v' \to v$ does not already exist), setting $I(v \to v') =]-\infty, +\infty[$. The propagation rule is local (Figure 4.16): let $e_0 = v \to v''$ be an edge, and let $e_1 = v \to v'$ and $e_2 = v' \to v''$ be an alternate path of length 2 relating v to v'' in G. One then updates

$I(e_0)$ by

$$I(e_0) := I(e_0) \cap [I(e_1) + I(e_2)]$$

or equivalently

$$\alpha(e_0) := max[\alpha(e_0), \alpha(e_1) + \alpha(e_2)]$$
$$\beta(e_0) := min[\beta(e_0), \beta(e_1) + \beta(e_2)]$$

If some $I(e_0)$ becomes empty, the chronicle is invalid or inconsistent: no date on events can satisfy these constraints. Moreover, the local consistency that the above propagation rule enforces can be proved to be equivalent to the global consistency. Therefore the 'if' is actually also an 'only if'. The propagation rule must be applied to every triple of edges (up to reversing some of them), which results in a complexity in $\mathcal{O}(|V|^3)$. Figure 4.17 illustrates a graph of temporal constraints before and after propagation of constraints.

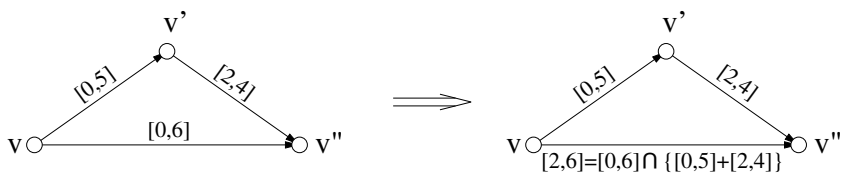


Figure 4.16: The constraint propagation rule in a chronicle.

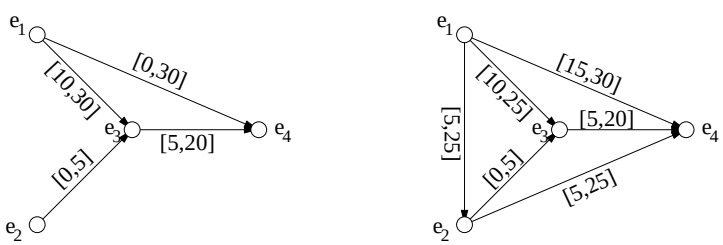


Figure 4.17: A time constraint graph before (left) and after (right) constraint propagation.

Recognition algorithm. The procedure maintains a set of partially recognized chronicles, i.e. partial mappings ϕ from G to the event set (Dousson, 1996; Dousson and Maigat, 2007). This is initiated as soon as a minimal event in the chronicle is identified in the flow. As this imposes a date to a vertex of G, the constraint propagation algorithm can be used to determine intervals for the arrival *dates* of the other events. If such an expected successive event appears in the flow, with correct type and correct time, the partial instance ϕ gives birth to two partial instances: an extension ϕ' of ϕ including the new observed event a, followed by a propagation of the constraints imposed by the date $\delta(a)$, and a copy of ϕ to prepare the ground for another event a' with the same type as a that may appear later.

A partially recognized chronicle may die if an expected event does not appear in time. To this end, it is assumed that events are not observed exactly in the order they are produced (so event dates may not be increasing in the observed sequence), but they reach the supervisor with some bounded delay. Alternately, one can feed partial instances of chronicles with special events that set the actual time, and which may reveal that not yet observed events will definitely be missing. When a partial instance is completed, it is displayed as a pattern of alarms, and the "diagnosis" corresponding to this chronicle is output.

Chronicle construction. As for all model-based approaches, the inference algorithms are well mastered and rather efficient, and the main difficulty lies in the identification of the relevant model. Here, this amounts to a) determining the relevant chronicles and b) associating them to some diagnosis. There are essentially two approaches: by model construction, or by learning. The first approach relies on a fault model of the network, and more specifically on a model of fault propagations. This brings us back to issues discussed in the section about causal graphs. In (Guerraz and Dousson, 2004), the authors rely on previous contributions (Aghasaryan et al., 1998) that modeled fault propagations in a network under the form of safe Petri nets. Without going too much into the details, safe Petri nets are dynamic systems that explicitly represent the parallelism of some events. In that sense, they contrast with automata that can only encode sequences of events. Safe Petri nets can be provided with a true concurrency (or partial order) semantics that represents their executions as partial order of events. It was shown in previous works of our team that distributed diagnosis procedures could be derived from a modeling of fault propagations as distributed Petri nets. (Guerraz and Dousson, 2004) proposes to extend this setting with timing, under the form of timed Petri nets. As a result, runs of these models immediately take the form of partially ordered transition firings related by time constraints, i.e. chronicles. The last step consists in selecting runs of the model that are exemplary of the failure situations one wishes to identify.

A second technique consists in directly examining the (huge) alarm logs collected by equipment in order to detect regularity patterns. Once such patterns are detected, they can be examined by experts that will then interpret them, label them with a diagnosis and a severity level, and select those that should be monitored. This semi-automatic data/process mining approach has the advantage of relying on easily available data, regardless of the presence or not of failures. It may yield normal behaviors, serious errors, but also some frequent malfunctions that fly below the radar (temporary or non critical problems). They also offer the advantage of automatically selecting the most frequent issues.

Chronicle learning is still an active topic in network management (see for example (Kavulya et al., 2012)). (Dousson and Duong, 1999; Fessant et al., 2005) proposed the FACE approach: Frequency Analysis for Chronicle Extraction. The algorithm is based on the obvious remark that any frequent chronicle in an alarm log is an assembling of frequent sub-chronicles. Therefore one can first look for frequent small chronicles and then try to combine them into larger ones. The method proceeds in two

steps: first the identification of the structure of a chronicle, then the setting of its time constraints. This is better illustrated by an example. Consider the following alarm log:

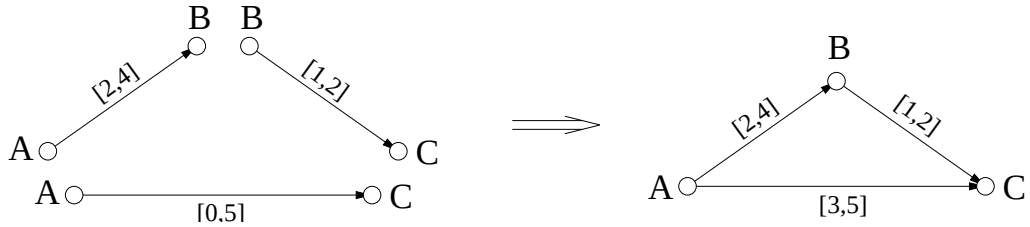


Figure 4.18: Merging 3 chronicles of size 2 into a chronicle of size 3, with time constraints propagation.

$$(A,1)\ \ (B,3)\ \ (D,4)\ \ (C,4)\ \ (A,4)\ \ (B,8)\ \ (C,9)\ \ (B,10)\ \ (B,12)$$

The first step consists in detecting unconstrained chronicle instances made of 2 events: all 10 possibilities are examined for pairs of event types in $\{A, B, C, D\}$, given that an unconstrained chronicle AB is of course equivalent to BA. A chronicle recognizer yields the following frequencies (for technical reasons, it is assumed that two chronicle instances can't share alarms, or equivalently that alarms are assigned to a unique instance):

$AA : 1$	$AB : 2$
$AD : 1$	$AC : 2$
$BD : 1$	$BB : 2$
$CC : 1$	$BC : 2$
$CD : 1$	
$DD : 0$	

The less frequent chronicles are discarded. For each of the most frequent ones (right-hand side column), one then observes the dates that appear in each detected instance to determine the time constrains of the chronicle. For example, from the two instances $\{(A,1),(B,3)\}$ and $\{(A,4),(B,8)\}$ one determines the time interval $[2,4]$ for edge $A \rightarrow B$. This interval computation must reflect the most relevant detected instances, and reject some obvious outliers. The recursion step then consists in merging 3 frequent chronicles of size 2 into a chronicle of size 3, for example AB, BC and AC into ABC (see Figure 4.18). This yields 4 possibilities here, from which one recomputes frequencies, selects the most frequent ones, adjusts time constraints, and so on.

$BBB : 1$	$ABB : 2$	$ABBC : 1$
	$ABC : 2$	
	$BBC : 2$	

In the end, one preserves the most frequent chronicles that were not assembled into a frequent larger one.

Discussion. The chronicle method belongs to pattern recognition approaches, and relies on rather efficient detection algorithms. They offer the nice feature of processing alarms on the fly, taking their date into account, which is a clean way to go beyond a plain reasoning based on a sliding time window. In the currently available settings, however, chronicles are sensitive to alarm losses, and there is no probabilistic version of chronices. Their explanation power is limited to the interpretation made by experts for each chronicle inserted into the recognition engine.

On the modeling side, chronicles can again be derived from fault propagation models, but most interestingly, there exist semi-automatic learning algorithms that can discover chronicles from alarm logs, an abundant resource which is rarely exploited. However, the unability of chronicles to adapt to changing networks, and the presence of an expert in the loop, remain important drawbacks.

4.8 Bayesian Networks

Bayesian networks are another possible choice for alarm correlation due to their ability to handle uncertainty and represent cause and effect relationships. A Bayesian Network (BN) is a directed acyclic graph (traditionally abbreviated DAG) whose nodes represent random variables, the edges denote existence of direct causal influences between the linked variables and the strengths of these influences are expressed by forward conditional probabilities (see Figure 4.19). In the context of fault diagnosis, the random variables represent states of network objects or the occurrence of network events. An

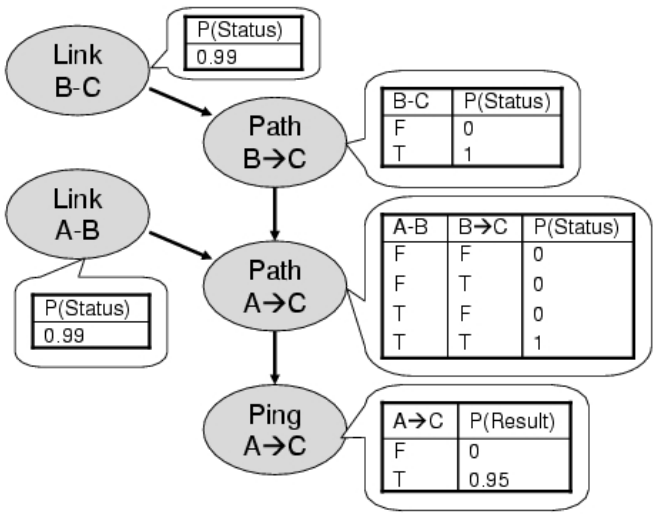


Figure 4.19: A Bayesian network for IP paths diagnosis

important advantage that Bayesian networks offer is the avoidance of building huge joint probability distribution tables that include permutations of all the nodes in the network. Rather, only the possible states of a node's immediate predecessors and their

effects on the node are necessary.

Definition 7 *A Bayesian network consists of the following:*

- *a set of variables and a set of directed edges between variables;*
- *each variable has a finite set of mutually exclusive states;*
- *the variables together with the directed edges form a DAG, a directed graph is acyclic if there is no directed path* $X_1 \to \cdots \to X_n$ *so that* $X_1 = X_n$*;*
- *to each variable* X *with parents* $W_1, ..., W_n$*, a conditional probability distribution (CPD)* $P(X|W_1, ..., W_n)$ *is attached; if* X *has no parents, then the table reduces to the unconditional probability table* $P(X)$*.*

Conditional probability distributions can be described in a variety of ways. A common, but not necessarily compact representation for a CPD is a table which contains a row for each possible set of values for the parents of the node describing the probability of different values for X. These are often referred to as *table CPDs* and are tables of multinomial distributions. Other possibilities are to represent the distributions via a tree structure (called, *tree-structured CPDs*), or using an even more compact representation such as a *noisy-OR* or *noisy-MAX*.

A Bayesian network is made up of various types of nodes. A *root* node has no parents, and a *leaf* node has no children. An *evidence* node, also known as a *finding node*, *instantiated node*, or *observed node* represents a variable with a single value that has been observed with probability one. Such a node is said to have been instantiated with *specific* or *hard evidence*. The set of all evidence variables is denoted $\boldsymbol{E}$ with values $\boldsymbol{e}$. The conditional and prior probabilities are part of the design of the network and do not change during inference. The dynamic values of the node probabilities shall be denoted $Bel(x)$. This latter value reflects the overall belief in the proposition $X = x$ given all the evidence, that is,

$$Bel(x) \triangleq p(x|\boldsymbol{e}) \tag{4.1}$$

As such, the belief values represent the posterior probabilities. The posterior probability of any variable may be obtained through marginalisation. A vector of belief values for all possible states of X is denoted $\boldsymbol{Bel}(X)$.

Belief propagation

A Bayesian network represents a complete probabilistic model of all the variables in a specific domain. Therefore, it contains all the information required to answer any probabilistic question about any variable in that domain. These questions are usually restricted to determining the most likely hypothesis or, more specifically, the belief in, or probability of, each possible hypothesis, given the available observations or evidence. The application of evidence is ongoing as the observations continue to arrive over a period of time. Therefore, the Bayesian network is dynamic and the most likely hypotheses, and their probabilities, are likely to change.

To ascertain the effect of the available evidence on any variable (hypothesis), each node updates the posterior probabilities for each of its hypotheses on the receipt of data messages from its immediate neighbours. The effect of evidence is transmitted throughout the network by allowing each variable to continually update its state by comparing the state of its neighbours against local constraints; if the local constraints are satisfied, no activity takes place, otherwise action is taken to correct the constraint violation and messages are sent to the immediate neighbours of that variable.

The posterior probability, or *belief*, for variable $X = x$ is $Bel(x) \triangleq p(x|\boldsymbol{e})$. The evidence influencing X is separated into two disjoint subsets, $\boldsymbol{e}_X^-$ denoting the evidence introduced through the arcs between X and its children, and $\boldsymbol{e}_X^+$ denoting the evidence introduced through the arcs between X and its parents. These may be further divided into $\boldsymbol{e}_{XY_i}^-$ for the evidence introduced through the arc to child Y_i, and $\boldsymbol{e}_{W_jX}^+$ as that introduced through the arc from parent W_j. Applying Bayes' rule conditioned on $\boldsymbol{e}_X^+$, and since $\boldsymbol{e}_X^-$ and $\boldsymbol{e}_X^+$ are independent given X, the belief may be written

$$Bel(x) = p(x|\boldsymbol{e}_X^-, \boldsymbol{e}_X^+) = \frac{p(\boldsymbol{e}_X^-|x, \boldsymbol{e}_X^+)\, p(x|\boldsymbol{e}_X^+)}{p(\boldsymbol{e}_X^-|\boldsymbol{e}_X^+)} = \frac{p(\boldsymbol{e}_X^-|x)\, p(x|\boldsymbol{e}_X^+)}{p(\boldsymbol{e}_X^-|\boldsymbol{e}_X^+)}. \tag{4.2}$$

Defining $\pi(x) = p(x|\boldsymbol{e}_X^+)$ and $\lambda(x) = p(\boldsymbol{e}_X^-|x)$, the belief becomes

$$Bel(x) = \alpha\, \pi(x)\, \lambda(x), \tag{4.3}$$

where α represents a normalizing constant.

Variable $\lambda(x)$ is the likelihood representing diagnostic or retrospective support for the proposition $X = x$. Vectors or messages $\boldsymbol{\lambda}(X)$, where each element of $\boldsymbol{\lambda}(X)$ corresponds to a different state of X, are passed up the network in the opposite direction to the arrows on the arcs i.e. from the children of X to X. Conversely, $\pi(x)$ may be considered as the prior certainty of x given the prior evidence $\boldsymbol{e}_X^+$, and it represents the causal or predictive support for $X = x$. Messages $\boldsymbol{\pi}(x)$, also representing each possible state of X, are passed down the network in the direction of the arrows i.e. from X to its children. For simplicity, it is assumed that all evidence is introduced into the network through leaf nodes, that is, all evidence on non-leaf nodes is represented by adding child leaf nodes.

The following description of the propagation in chains, trees and polytrees is taken from (Krieg, 2001). Note that the product $\boldsymbol{F}(X)\, \boldsymbol{G}(X)$ of two such vectors will stand for term-by-term multiplication while the dot symbol $\cdot$ will be used to indicate matrix products.

Propagation in Chains

Consider the serial or chain network in Figure 4.20, where $\boldsymbol{\lambda}(Y) = p(\boldsymbol{e}_Y^-|Y)$. Now,

$$\lambda(x) = p(\boldsymbol{e}_X^-|x) = \sum_y p(y|x)\, p(\boldsymbol{e}_Y^-|y) \Rightarrow \boldsymbol{\lambda}(X) = \boldsymbol{\lambda}(Y) \cdot p(Y|X). \tag{4.4}$$

Similarly, $\boldsymbol{\lambda}(W) = \boldsymbol{\lambda}(X) \cdot p(X|W)$. Note that $\boldsymbol{\lambda}(X)$ is calculated from the probability

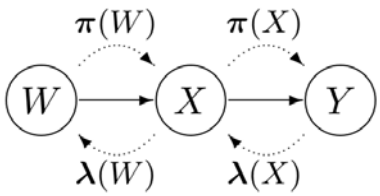


Figure 4.20: Propagation in a chain network (taken from (Krieg, 2001))

table $p(Y|X)$ attached to node Y and is calculated at node Y. To complete the belief propagation, consider the propagation of the $\boldsymbol{\pi}$ messages, starting at node W. Node W is a root node and its only *evidence* is its prior probability distribution. Therefore, $\pi(w) = p(w|\boldsymbol{e}_W^+) = p(w)$, and

$$\pi(x) = p(x|\boldsymbol{e}_X^+) = \sum_w p(x|w)p(w|e_W^+) \Rightarrow \boldsymbol{\pi}(X) = \boldsymbol{\pi}(W) \cdot p(X|W). \tag{4.5}$$

Similarly, $\boldsymbol{\pi}(Y) = \boldsymbol{\pi}(X) \cdot p(Y|X)$. The $\boldsymbol{\pi}$ messages are propagated as illustrated in Figure 4.20, and the belief at each node can be calculated using (4.3). The appropriate $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ messages are re-propagated if evidence is added or changed.

Propagation in Trees

Now, consider a case of multiple child nodes, as illustrated in Figure 4.21. The additional problems introduced here are combining the $\boldsymbol{\lambda}$s as they are propagated up the tree and separating the $\boldsymbol{\pi}$s as they are propagated down the tree. Starting with the

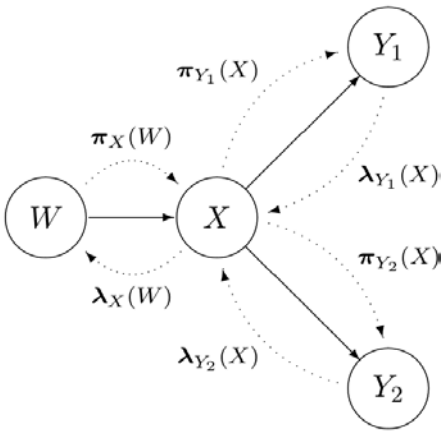


Figure 4.21: Propagation in a tree (taken from (Krieg, 2001))

$\boldsymbol{\lambda}$s, the evidence on the sub-tree at X is partitioned into the disjoint sets $\boldsymbol{e}_{Y_1}^-$ and $\boldsymbol{e}_{Y_2}^-$.

Therefore,

$$\lambda(x) = p(\boldsymbol{e}_{Y_1}^{-}|x)\, p(\boldsymbol{e}_{Y_2}^{-}|x) \triangleq \lambda_{Y_1}(x)\, \lambda_{Y_2}(x) \Rightarrow \boldsymbol{\lambda}(X) = \boldsymbol{\lambda}_{Y_1}(X)\, \boldsymbol{\lambda}_{Y_2}(X). \tag{4.6}$$

Node X then transmits $\boldsymbol{\lambda}_X(W) = \boldsymbol{\lambda}(X) \cdot p(X|W)$ to node W. Assuming we have $\boldsymbol{\pi}(X) = \boldsymbol{\pi}_X(W) \cdot p(X|W)$, it is now necessary to split $\boldsymbol{\pi}(X)$ into individual messages for each child of X, that is Y_1 and Y_2. Consider the predictive support for $X = x$ that is passed to node Y_1, namely,

$$\pi_{Y_1}(x) = p(x|\boldsymbol{e}_X^{+}, \boldsymbol{e}_{Y_2}^{-}). \tag{4.7}$$

Using Bayes' rule,

$$\pi_{Y_1}(x) = \alpha\, p(\boldsymbol{e}_{Y_2}^{-}|x) p(x|\boldsymbol{e}_X^{+}) = \alpha\, \lambda_{Y_2}(x)\pi(x). \tag{4.8}$$

More generally, for the kth child of K children of X,

$$\pi_{Y_k}(x) = \alpha \prod_{j=1\backslash k}^{K} \lambda_{Y_j}(x)\pi(x) = \alpha\, \frac{Bel(x)}{\lambda_{Y_k}(x)} \Rightarrow \boldsymbol{\pi}_{Y_k}(X) = \alpha\, \frac{\boldsymbol{Bel}(X)}{\boldsymbol{\lambda}_{Y_k}(X)}, \tag{4.9}$$

allowing the same message to be passed to all children, namely $\boldsymbol{Bel}(X)$, for determination of $\pi_k(X)$ at each child.

Propagation in Polytrees

Causal polytrees are singly connected trees in which each node may have multiple parents. They may be thought of as collections of causal trees, fused at the nodes where the arrows converge. To illustrate the propagation in this type of network,

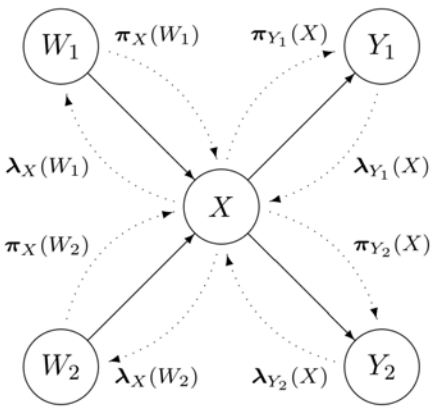


Figure 4.22: Propagation in a causal polytree network

consider the example in Figure 4.22. The additional problems introduced by this type of structure are combining the $\boldsymbol{\pi}$ messages from multiple parent nodes and splitting

the $\boldsymbol{\lambda}$ messages between multiple parent nodes. Consider the variable X that has two parents, W_1 and W_2, and two children, Y_1 and Y_2. The evidence obtained from the sub-tree rooted at X is the union of evidence obtained from each arc connecting it to its children, that is, $e^-_{XY_1}$ and $e^-_{XY_2}$. Similarly, the evidence from the rest of the network can be separated into that obtained from each arc linking X to each of its parents, namely $\boldsymbol{e}^+_{W_1X}$ and $\boldsymbol{e}^+_{W_2X}$ (Pearl, 1988).

The $\boldsymbol{\lambda}$ messages, $\boldsymbol{\lambda}(X) = \boldsymbol{\lambda}_{Y_1}(X)\,\boldsymbol{\lambda}_{Y_2}(X)$ must be split for transmission to the parents of X, namely $\boldsymbol{\lambda}_X(W_1)$ for W_1 and $\boldsymbol{\lambda}_X(W_2)$ for W_2. Therefore, we denote $\lambda_X(w_1) = p(\boldsymbol{e}^-_{W_1X}|w_1)$ and separate the evidence $\boldsymbol{e}^-_{W_1X}$ into disjoint sets, namely that associated with the sub-tree rooted at X, $\boldsymbol{e}^-_X$, and that associated with all other parents of X, in this case $\boldsymbol{e}^+_{W_2X}$. Then,

$$\lambda_X(w_1) = p(\boldsymbol{e}^-_{W_1X}|w_1) = p(\boldsymbol{e}^-_X, \boldsymbol{e}^+_{W_2X}|w_1), \tag{4.10}$$

and conditioning on X and W_2 and applying Bayes' Rule gives

$$\lambda_X(w_1) = \beta \sum_x p(\boldsymbol{e}^-_x|x) \sum_{w_2} p(w_2|\boldsymbol{e}^+_{W_2X}) p(x|w_1, w_2), \tag{4.11}$$

where β is an arbitrary constant. In general, for n parents:

$$\lambda_X(w_i) = \beta \sum_x \lambda(x) \sum_{w_k:k\neq i} p(x|w_1, w_2, ..., w_n) \prod_{k=1\backslash i}^{n} \pi_X w_k, \tag{4.12}$$

where $\pi_X(w_k) = p(w_k|\boldsymbol{e}^+_{W_kX})$. The predictive support is provided through the arcs from the parents of X, that is, $\pi(x) = p(x|\boldsymbol{e}^+_X) = p(c|\boldsymbol{e}^+_{W_1X}, \boldsymbol{e}^+_{W_2X})$. Conditioning on W_1 and W_2, this becomes

$$\pi(x) = \sum_{w_1} \sum_{w_2} p(x|w_1, w_2) \pi_X(w_1) \pi_X(w_2). \tag{4.13}$$

In general, for n parents, this may be written,

$$\pi(x) = \sum_{w_1, w_2, ..., w_n} p(x|w_1, w_2, ..., w_n) \prod_{i=1}^{n} \pi_X(w_i). \tag{4.14}$$

Application of Bayesian Networks for fault management

Several applications of Bayesian networks for fault management have been reported. (Deng et al., 1993) investigated the application of probabilistic reasoning to fault diagnosis in Linear Lightwave Networks (LLN's). The basic network components are LDC's (linear divider/combiner). The observable messages are the LDC input and output power values. If an LDC has failed or some part of it has failed, all its output power values will be incorrect (outside the specified range) but its input power values remain unaffected. The LLN routes/connections determine the internal connections

between the input and the output ports of every LDC. The inference model is constructed as follows. A power value, say F_{ip}, is assumed to be in one of the two states, within the expected range or outside it. A binary-valued random variable F_i is defined for power value F_{ip}: $F_i = 1$ if the power value F_{ip} is within the expected range, and 0 otherwise. Similarly, for each LDC a binary-valued random variable L_j is defined: $L_j = 1$ if LDC_j is under normal operation, and 0 otherwise. Furthermore, the L_j's are assumed to be mutually independent. The dependence relations among these random variables can be graphically represented as a DAG. For a non-root random variable X in the DAG, X and its parents $U_1, ..., U_n$, are related by $X = U_1 \wedge U_2 \wedge ... \wedge U_n$. The conditional probability $P(x|u_1, ..., u_n) = 1$ if $x = u_1 \wedge ... \wedge u_n$ and 0 otherwise. By the DAG and the defined probability distributions, the LLN inference model is represented by a Bayesian network. An inference algorithm capable of conducting fault diagnosis with incomplete evidence and on an interactive basis is then proposed.

(Steinder and Sethi, 2002) proposed a layered system model that represents relationships between services and functions offered between neighboring protocol layers. These relationships form a bipartite probabilistic dependency graph where each node is associated with multiple failure modes $F_1, ..., F_k$, which represent availability and performance problems pertaining to the service or function represented by the node. The chosen BN structure represents a model of conditional probabilities known as noisy-OR gates. This simplified model contains binary-valued random variables. A mapping from the above dependency graph to the BN is performed as follows. For every node of the dependency graph and for every failure mode associated with this node, a random variable is created, whose domain is $\{true, false\}$ (1 and 0 are also used to represent values $true$ and $false$, respectively). Let V_i be a belief network node created for failure mode F_j of the dependency graph node representing a service or a function. Assignment $V_i = true$ ($Vi = false$) indicates that the service/the function is (is NOT) in condition F_j. For every dependency graph edge $X \rightarrow Y$ and for every failure mode of node Y,F_i, the corresponding failure mode of node X, F_j is determined. Let V_i be the belief network node corresponding to Y and F_i. Let V_j be the belief network node corresponding to X and F_j. A belief network edge is inserted from V_i to V_j. A probability matrix associated with dependency link $X \rightarrow Y$. A symptom is defined as an observation that a dependency graph node X is in condition F_j (negative symptom), or is NOT in condition F_j (positive symptom). If V_i is the belief network node corresponding to X and its failure mode F_i, then the negative symptom and positive symptom are interpreted as an instantiation of V_i with value *true* and *false*, respectively. As a result of this mapping, the set of all observed symptoms becomes the evidence set e. The problem of finding the subset of (all possible) faults is equivalent to computing the MPE query based on the evidence set e.

(Steinder and Sethi, 2004b) presents a probabilistic event-driven fault localization technique, which uses the so-called probabilistic symptom-fault map as a fault propagation model. With every fault $f_i \in \mathcal{F}$ a probability of its independent failure is associated, which is denoted by $p(f_i)$. The edge between $f_i \in \mathcal{F}$ and $s_j \in \mathcal{S}$ indicates that f_i may cause s_j. The edge is weighted with the probability of the causal implication, $p(s_j|f_i)$. A noisy-OR model of probability distribution is considered. A

subset of symptoms observed is denoted by $\mathcal{S}_{\mathcal{O}}$. The purpose of fault localization is to find $\mathcal{F}_{\mathcal{D}} \subseteq \mathcal{F}$ that maximizes the probability that (1) all faults in $\mathcal{F}_{\mathcal{D}}$ occur and (2) each symptom in $\mathcal{S}_{\mathcal{O}}$ is explained by at least one fault from $\mathcal{F}_{\mathcal{D}}$. A fault localization technique called Incremental Hypothesis Updating (IHU) is proposed. Its basic version creates a set of hypotheses, each of which is a subset of $\mathcal{F}$ that explains all symptoms in $\mathcal{S}_{\mathcal{O}}$. Hypothesis $h_j \subseteq \mathcal{F}$ explains symptom $s_i \in \mathcal{S}_{\mathcal{O}}$ if it contains at least one fault that explains s_i. The hypotheses are ranked using a belief metric, b, which expresses the confidence associated with a given hypothesis relative to other hypotheses. The algorithm is triggered by an observation of the ith symptom, s_i, and creates a set of hypotheses, $\mathcal{H}_i$, each explaining symptoms s_1 through s_i. Set $\mathcal{H}_i$ is created by updating $\mathcal{H}_{i-1}$ with an explanation of symptom s_i. Set H_{s_i} is defined as a set $\{f_k \in \mathcal{F}\}$ such that f_k may cause s_i. To incorporate the explanation of symptom s_i into a set of fault hypotheses, in the ith iteration of the algorithm, each $h_j \in \mathcal{H}_{i-1}$ is analysed. If h_j is able to explain symptom s_i, h_j is put into $\mathcal{H}_i$. Otherwise, h_j has to be extended by adding to it a fault from H_{s_i}. Fault $f_l \in H_{s_i}$ may be added to $h_j \in \mathcal{H}_{i-1}$ only if the size of h_j, $|h_j|$, is smaller than $\mu(f_l)$, defined as the minimal size of a hypothesis in $\mathcal{H}_{i-1}$ that contains f_l and explains symptom s_i. The proposed solution allows multiple simultaneous independent faults to be identified and is then extended to deal with loss and spurious symptoms.

(Khanafer et al., 2008) presented an automated diagnosis system for UMTS networks using Bayesian Network approach. Two components of the automatic diagnosis system have been distinguished, i.e. the diagnosis model and the inference method. The elements of the model are causes and symptoms. Causes are modeled as discrete random variables with two states (*absent / present*). Two types of symptoms are considered: Alarms and KPIs. Alarms are also modeled as discrete random variables with two states (*on / off*). KPIs can be modeled as discrete random variables with two, three, or more states, each representing a subset of the continuous range of the KPI. The chosen BN structure is a so-called Naive Bayes Model (NBM), which is also known as a Simple Bayes Model or Naive Bayesian Classifier. The NBM (Figure 4.23) consists of a parent node C, whose states are the possible fault causes, and the children nodes $S_1, ..., S_N$, which represent the symptoms and may have any discrete number of states. This model assumes that there can only be one fault happening

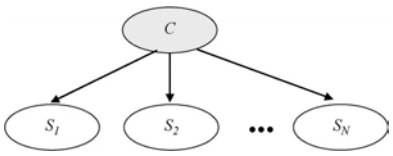


Figure 4.23: NBM

at the same time. The impact of this simplification on the diagnosis performance is considered minor while the simplicity of this type of model is a clear benefit. The required probabilities for this model are the following: $p(C = c_k)$ for each fault cause c_k;

$p(S_i = s_{i,j}|C = c_k)$ for each state $s_{i,j}$ of S_i and each fault cause c_k. Thus, with the NBM structure, the desired conditional probabilities of the fault causes given the observed symptoms can be calculated as

$$p(C = c_k|E) = \frac{p(C = c_k) \prod_{i=1}^{N} p(S_i = s_{i,j}|C = c_k)}{p(E)}.$$

This equations assumes that the symptoms S_i are independent given the fault cause C. According to the independence properties of a BN, this is exactly what the NBM structure indicates. To further enhance automation and performance of the BN model, the automated learning of both KPI thresholds and model probabilities has been thouroughly investigated.

Discussion Due to Bayesian network's form of knowledge representation, large amounts of interconnected and causally linked data can be represented. Generally speaking: Bayesian networks can represent deep knowledge by modeling the functionality of the transmission network in terms of cause and effect relationships between element and network behavior and faults.

Bayesian networks can provide guidance in diagnosis. Calculations over the same BN can determine both the precedence of alarms and the areas that need further clarification in order to provide a finer grained diagnosis. They can handle noisy, transient, and ambiguous data due to their grounding in probability theory. They have a modular, compact, and easy to understand representation, when compared to other probabilistic methods. They provide a compact and well-defined problem space because they use an exact solution method for any combination of evidence or set of faults.

Bayesian networks are appropriate for automated diagnosis because of their deep representations and precise calculations. A concise and direct way to represent a system's diagnostic model is as a Bayesian Network constructed from relationships between failure symptoms and underlying problems. A Bayesian Network represents cause and effect between observable symptoms and the unobserved problems so that when a set of symptoms are observed the problems most likely to be the cause can be determined. In practice, the network is built from descriptions of the likely effects for a chosen fault. In use as a diagnostic tool, the system reasons from effects back to causes.

The difference of Bayesian Networks, in comparison with other classical methods, is their polyvalence. They allow dealing with issues such as prediction or diagnosis, optimization, data analysis of feedback experience, deviation detection and model updating. The graphical representation is interesting since the model complexity is understandable in a single view. In the case of large size model, object oriented representation OOBN or probabilistic relational descriptions (PRM) provide manageable models.

The development of a diagnostic Bayesian Network requires a deep understanding of the cause and effect relationships in a domain, provided by domain experts. This is both an advantage and disadvantage. An advantage is the knowledge is not represented as a black box, as are the Neural Networks. Thus, humanly understandable explanations of diagnoses can be given. The disadvantage is that the realm of the con-

sidered system/network can be technologically immature and as a result expertise in fault diagnosis may be hard to find and to implement.

4.9 Summary

Many approaches to fault management rely on black-box or learning methods (neural networks, self-organizing maps, statistical learning (Kavulya et al., 2011), codebook techniques (Reali and Monacelli, 2009), chronicle discovery (Dousson, 1996; Dousson and Duong, 1999; Kavulya et al., 2012)). Such strategies generally suffer from a lack of classified data relating failures to some observed symptoms. They also poorly resist to phenomena like complex symptom patterns due to multiple faults, or to the asynchronism or loss of observations, and they are difficult to maintain as the network evolves. Such approaches are thus appropriate for simple and small scale event correlation over fixed topologies, where training can safely converge.

To progress towards the more ambitious objectives mentioned above, one must adopt a model-based strategy, which consists first in building a model of the managed network, establishing relations between correct/incorrect functioning and observed signals/symptoms, and then in deriving event management algorithms based on this model. This is the track adopted by several contributions (see the 'model traversal' techniques in (Steinder and Sethi, 2004a), and their generalization into 'graph theoretic' techniques). One trend compiles expert knowledge about malfunctions and their consequences, or symptoms and their causes, into causal graphs that form the support of automatic (possibly distributed) reasoning (Lu et al., 2011) and (Grosclaude, 2008) (in French). Another important trend models or learns from data the (logical or statistical) dependencies between resources, initial faults and observable symptoms, and compile them into Bayesian networks or related objects (Bouloutas et al., 1994, 1995; Fabre et al., 2004). The inference engines on such structures can then be taken off the shelf, since this is a well understood and much covered topic, and the distribution of these techniques is also a well paved trail (Fabre et al., 2004, 2005). As witnessed by the above contributions, however, model-based techniques always raise two major difficulties: a) how to derive such a model, suited to a given network at a given time, in particular if one wishes to capture several network layers and segments (see open problems 1, 4 and 6 in (Steinder and Sethi, 2004a)) and b) how to reason on a potentially huge model, if one wishes to manage a nation-wide network for example. This thesis proposes a contribution to these two issues.

Obtaining a model of the managed network is far from simple, especially if one wishes to capture inter-layer and inter-segment fault propagations. The first information source one should use is of course the topology of the network, which many contributions considered (for e.g. (Bouloutas et al., 1994)), including professional tools dedicated to specific network technologies. But one should go further and aggregate different sources of information, generally found in the standards, in protocol descriptions, and in expert knowledge if available. Relying on the past experience of our team in the modeling of fault and alarm propagations (Fabre et al., 2004), we propose

here a self-modeling approach. This consists first in identifying the different families of network resources that should be managed, and how these resources are structured and rely on each other, following the usual hierarchical construction of networks. This yields a collection of generic patterns of resources and their dependencies, designed according to a specific grammar. Secondly, by exploring the managed network, one can then create as many instances of these generic patterns as discovered in the network topology. As these instances share some resources, this construction results in a large scale structure where similar patterns are duplicated and partially overlap. The model obtained in that way perfectly matches a given network, and can be used for diagnosis, root-cause analysis, and possibly for failure impact analysis.

Regarding the diagnosis engine, we propose to translate the model of network resources and their dependencies into the Bayesian network (BN) formalism, which seems to reach some consensus in the network management community. Bayes nets can easily mix statistical dependencies and constraints/logics, while enabling limited statistical learning (parameter identification) when data are available. They can also accommodate networks of dynamic systems (Fabre, 2007; Fabre et al., 2004; Fabre and Hadjicostis, 2006). Probabilistic reasoning on BN is well understood, and allows one to relate observations or test results to the state of hidden variables of interest. But dealing with possibly huge models may make them inappropriate. Therefore we propose to adapt the BN formalism to explore only part of the model, starting by the resources involved in a given malfunction to be explained, and progressively introducing/revealing more resources (*i.e.* variables) to reach new observations and making new tests, in order to locate the origin of the malfunction with more precision. Extra extensions would also be necessary, in particular to deal with the intrinsic hierarchical description of networks, but they will be examined in a forthcoming work.

As a support to these two research directions, the thesis considers the management of failures in IMS networks, capturing several network segments (access, metropolitan and core), and the end-to-end services running on top of them (Bertin et al., 2007).

Chapter 5

Structure of IMS networks

Contents

5.1 Physical layer of the IMS Architecture 89
5.2 Functional layer of the IMS architecture 90
5.2.1 Core IMS subsystem 91
5.2.2 RACS subsystem 95
5.2.3 NASS subsystem 97
5.3 Procedural layer of the IMS Architecture 101
5.3.1 NASS procedures 102
5.3.2 RACS procedures 104
5.3.3 Core IMS procedures 105
5.4 Contributions: Relevant structural properties 111

Starting from standards descriptions of IMS networks, our objective in this chapter is to identify the network resources involved in such networks, their structuring and above all their dependencies, which will be the knowledge used to diagnose malfunctions. The main idea is that the failure of a resource is either spontaneous, or results from the failure of a second resource that is necessary to the first one. The term 'resource' covers both physical equipment in charge of transport, pieces of software running over them, but also procedures (that must be completed in order to access IMS services). We represent an IMS network by a dependency model of three multi resolution layers.

5.1 Physical layer of the IMS Architecture

The physical layer[1] displayed in Figure 5.1 comprises an access network and a core network. The access network is made of two segments: the metro-access, and the metro-core. The former itself decomposes into smaller segments: access (first/last mile) and aggregation. The first mile stretches from the Customer Premise Equipment (CPE) up to the Access Node (AN). As for the aggregation segment, it aggregates

[1]The one considered here corresponds to a possible realization of the TISPAN NGN IMS functional architecture (TISPAN, 2009) with an xDSL-based access network.

traffic from different users, located behind different ANs, and connects them to the metro-core segment. Aggregation and metro-core segments are connected via one or more IP edge routers. In configurations where the access segment uses the Digital Subscriber Line (DSL) technology, the AN is known as a DSL Access Multiplexer (DSLAM) and the IP Edge router as a BRAS. The metro-core segment is connected to the core network, containing the IMS service platform, via routers such as the SBC. The Point of Presence (PoP) of Internet (IP) is generally at the SBC, while routing in the access network mostly uses Ethernet. Notice that the internal components of the core network and of the metro aggregation have not been detailed.

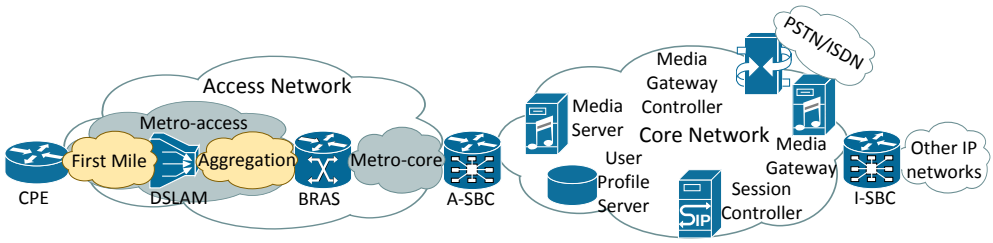


Figure 5.1: The generic IMS physical architecture

Properties of the physical layer

The physical layer architecture is made of specific equipment connecting various network segments. Such a connection is modelled by a 'connected-to' relationship between an equipment and a segment. Besides, the physical layer architecture is essentially hierarchical: network segments being decomposable into other physical equipment connecting smaller network segments.

5.2 Functional layer of the IMS architecture

IMS is defined in the form of a reference architecture to enable delivery of next-generation communication services over an IP network. It is considered a subsystem because it exists as part of a complete network. It is considered a reference architecture, because the implementers build the functional elements conforming to these specifications. The IMS architecture is defined in terms of functional elements, their interaction, which is termed as reference points, and the protocols that carry out these interactions. All functional elements within the IMS architecture communicate with IP-based protocols. Session Initiation Protocol (SIP) (Rosenberg et al., 2002) is the most prominent, as it provides the capability to establish and control multimedia sessions. One reason why SIP is important, is that IMS is seen to provide a uniform protocol for signaling with the endpoints. Besides SIP, other protocols support vital functions as well. Diameter (Calhoun et al., 2003) is the enabler for subscriber, policy, and charging functions. Megaco/H.248 (Cuervo et al., 2000; ITU-T, 2013) and Real-Time Protocol

(RTP) (Schulzrinne et al., 2003) provide the media-related support. Common Open Policy Service (COPS) (Durham et al., 2000) was used in the earlier IMS releases for policy functions, but has now given way to Diameter.

We now look at some of the functions provided by the various elements in the architecture. The functional architecture is made of three main subsystems. The NASS (TISPAN, 2010a) provides registration at the access level (identification and authentication) and initialization of the User Equipment (UE) (in particular IP address allocation) for accessing to the multimedia services. The RACS (TISPAN, 2006) is in charge of policy control, in particular resource reservation and admission control in the access and aggregation segments of the network. Finally the Core IP Multimedia Subsystem (TISPAN, 2007) (core IMS) is in charge of session initiation, provisioning and termination (SIP) for multimedia services directed to terminal users. Subsystems are hierarchical objects, comprising several functions (see below) that communicate with each other via interfaces (reference points).

5.2.1 Core IMS subsystem

The Core IP Multimedia Subsystem (TISPAN, 2007, 2009) displayed in Figure 5.2 comprises the the following functions.

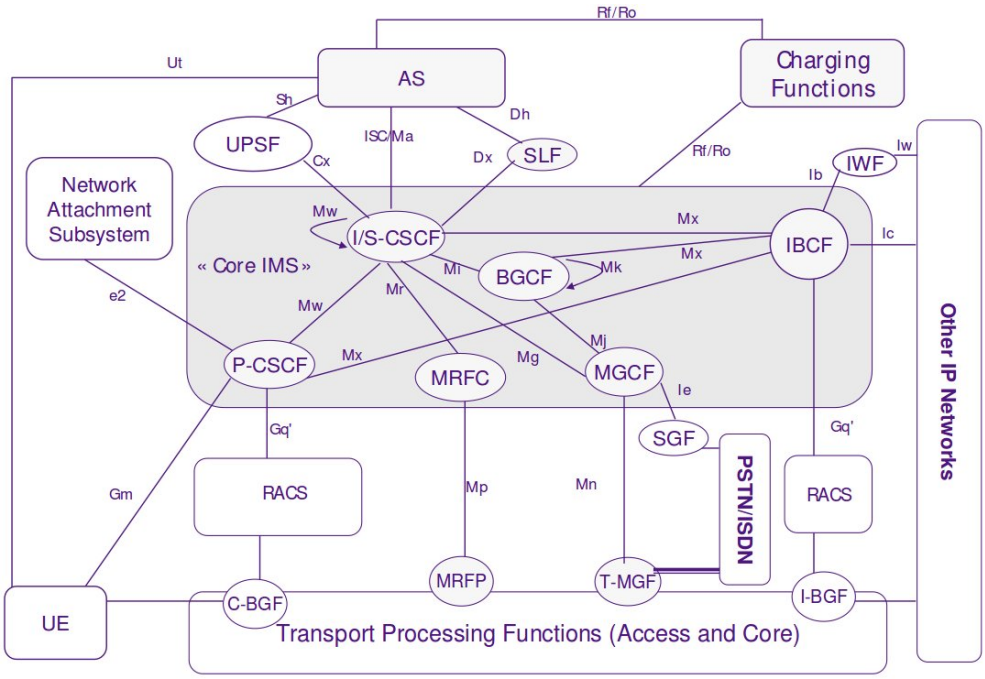


Figure 5.2: Core IP Multimedia Subsystem architecture (taken from (TISPAN, 2007))

Session Control Functions

The Call Session Control Function (CSCF) provides the central control function in the IMS Core Network to set up, establish, modify, and tear down multimedia sessions (Figure 5.3). The CSCF function is distributed across three types of functional elements based on the specialized function they perform. These three elements are the Proxy CSCF (P-CSCF), Interrogating CSCF (I-CSCF), and the Serving CSCF (S-CSCF).

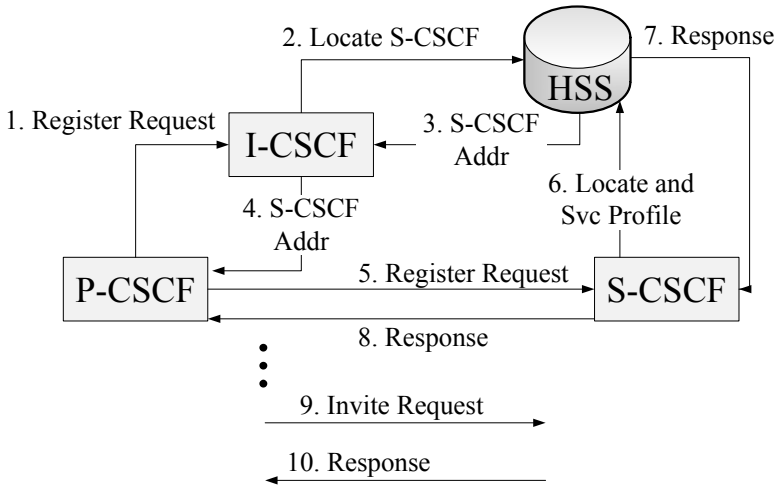


Figure 5.3: Session Control

The P-CSCF is an edge access function and is the entry point for a UE to request services from an IMS network. The role of this CSCF is to function as a proxy by accepting incoming requests and forwarding them to the entity that can service them. The incoming requests are either the initial registration or an invitation for a multimedia session. A request for the UE to register for a service is normally forwarded to a session controller or to one with the capability to interrogate for it. Requests that are a session invitation are directed by the P-CSCF to a serving CSCF. The P-CSCF provides a policy function by initiating support for IP flow control and authorization of traffic-bearer resources.

The I-CSCF is responsible for determining which serving CSCF should be assigned for controlling the session requested by the UE. A request to the I-CSCF may come from the home network or a visited network through the proxy CSCF. The I-CSCF obtains the request for the address of the S-CSCF from the User Profile Server Function (UPSF) during a registration request, and provides it to the P-CSCF for subsequent multimedia requests.

The S-CSCF is responsible for conducting both registration and session control for the registered UE's sessions. It functions as a registrar and enables the network location information of the UE to be available at the UPSF. It makes a determination to allow or deny service to the UE. It enables the assignment of application servers

to the session, if required. Its role is to execute the session request by locating the destination endpoint and conducting the signaling toward it.

All three CSCF functions are responsible for generating the Session Details or the Call Detail Records. The interface to the P-CSCF from the UE is the Gm interface that carries SIP or Session Description Protocol (SDP) signaling. The P-CSCF communicates to the interrogating and serving CSCF using SIP over the Mw interface.

Subscription Functions

The User Profile Server Function (UPSF) is a large database containing the complete subscription information about an IMS user. This information is accessible to all the IMS core elements needing information about the subscriber's profile, subscribed services, or authentication data. The UPSF supports the CSCF functions by: a) identifying the address of the CSCF that should be handling the session; b) storing the user's registration and location information; c) supporting the authentication and authorization by providing the integrity and ciphering data; d) and providing an access to a service profile, for which the subscriber has been provisioned. The UPSF also extends functionality to the application servers to determine service authorization, and also grants the capability to update subscriber profile data to application servers with provisioning capability. The CSCF communicates with the UPSF using Diameter over the Cx interface. The application servers use the Diameter Sh interface. A large network may require provisioning the subscribers set into more than one UPSF. This requires an intelligent entity to guide the requested CSCF or application server to the right UPSF. The Subscriber Locator Function (SLF) provides this support to the I-CSCF, the S-CSCF, and the application servers as well. The CSCFs request the UPSF determination from the SLF over the Diameter Dx interface. The application servers use the Diameter Dh interface.

Media Functions

Having examined the elements in the control plane, we now examine the functional elements responsible for processing the multimedia stream in the media plane (Figure 5.4). The Multimedia Resource Function (MRF) encompasses the functionality to control the media stream and provide resources for processing it. The MRF comprises the Multimedia Resource Function Controller (MRFC) and the Multimedia Resource Function Processor (MRFP). The bearer represents the actual multimedia stream carrying voice, data, and video. The MRFP provides the control of the bearer, which in the IMS core network is an RTP stream. It provides the necessary resources for processing the media stream. To support multimedia conferencing, it provides the capability to mix multiple media streams and manage access to shared resources.

The function of the MRFC is to control the resource pool of the MRFP. The MRFC and MRFP have a master-slave relationship. The MRFC controls the MRFP with an H.248 model over the Mp interface. The MRFC accepts the requests from the serving CSCF or an Application Server and controls the resources for the media stream

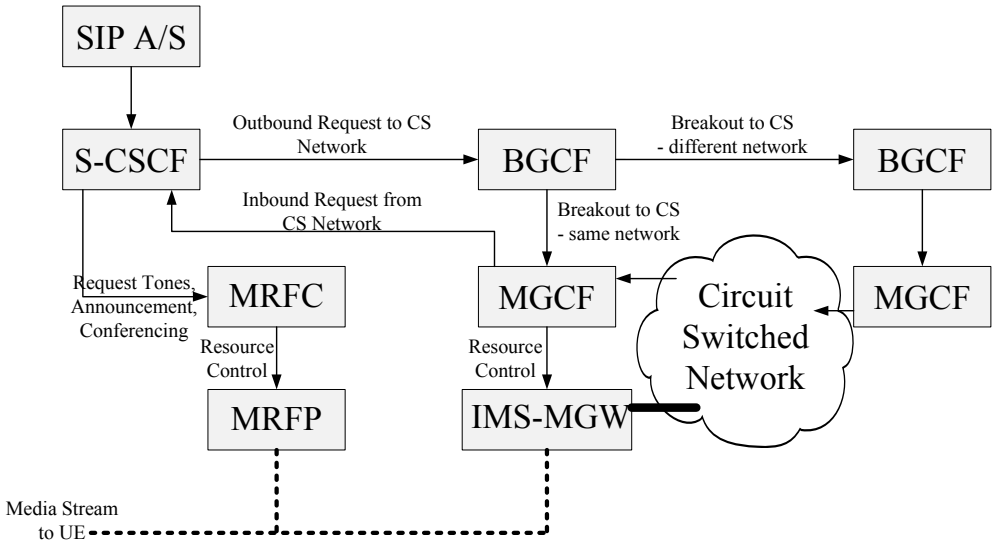


Figure 5.4: The media functions

accordingly. The serving CSCF and the application servers can request media resources and services using SIP over the Mr interface.

The second set of elements in the media plane provides the capability to inter-work with the legacy Circuit Switched (CS) network. Traffic is carried out on Time Division Multiplexing (TDM) streams. Media conversion between TDM and IP is performed by an IMS Media Gateway (IMS-MGW). The control function has functionality split into a Breakout Gateway Control Function (BGCF) and a Media Gateway Control Function (MGCF) based on whether a call has to go outbound to the CS network or it is an inbound call from the CS network. Similar to the MRFP, it provides the necessary Digital Signal Processing (DSP) resources for this function. In addition, it has to provide processing of the payload, which involves codecs, echo cancellation, and a bridge for conferencing. The MGCF, similar to the MRFC, provides control of the MGW resources. In addition, it has to handle an inbound call from a CS network. It therefore needs to identify the right serving CSCF based on the routing number of the incoming call. The MGCF is also responsible for protocol conversion between the CS network ISDN User Part (ISUP) signaling and SIP.

The serving CSCF requests the BCGF for determining, from the IMS network, which CS network the call needs to be directed to. If the CS network is managed within the same operator domain, the BGCF will direct the request to the MGCF. Otherwise, the BGCF will forward the signaling to the BGCF in that network. The S-CSCF communicates with the BGCF using SIP over the Mi interface. The BCGF communicates with the MGCF using SIP over the Mj interface. The BGCF also communicates with a BGCF in another IMS network over the Mk interface. The MGCF controls the IMS-MGW with Megaco/H.248 over the Mp interface.

Service Functions

The service plane in the IMS is designed to support the next generation of application with SIP, and to be able to work with the Legacy service platforms. The service plane elements, referred to as the application servers, have the capability to support full service logic for an application. They can additionally function as a gateway or provide the inter-working function to a legacy server or a non-SIP server. Or the elements could coordinate service logic between multiple servers.

Regardless of their role, all types of service elements communicate with the S-CSCF using SIP on the IMS Service Control (ISC) interface. They also have access to the subscriber information stored in the UPSF. The SIP Application Server (AS) is a SIP server platform providing the value-added service logic to the IMS session control. The SIP AS can support services for call control, presence, and messaging to name a few. The SIP AS can also be used to inter-work with a non-SIP service, such as Web-based services.

Border Functions

The Signaling Gateway (SGW) is a border function that provides the signaling conversion between IP-based protocols and the legacy SS7 networks. The Inter Working Function (IWF) provides the necessary support between different SIP profiles and between SIP and H323 systems. The Interconnection Border Control Function (IBCF) provides the edge function specific to the service provider network. It supports Session Border Control, interfacing for bandwidth control with RACS and invokes the IWF when necessary.

5.2.2 RACS subsystem

RACS subsystem (see Figure 5.5) provides

- Admission control. Performs admission of Quality of Service (QoS) resource requests based on the user profile, operator-specific policy rules, and resource reservation provided by NASS.

- Resource reservation. Verifies if the QoS resource request is within the permitted bandwidth in the access network, and reserves the resource.

- Gate control. Provides Network Address Port Translation (NAPT) control and priority traffic control, and performs gate control of the edge router based on the approved QoS resource request.

The elements in the RACS use Diameter to communicate the policy-related information. The RACS provides the support for guaranteed QoS by resource reservation. It also supports relative QoS with diffserv. RACS consists of the following function blocks:

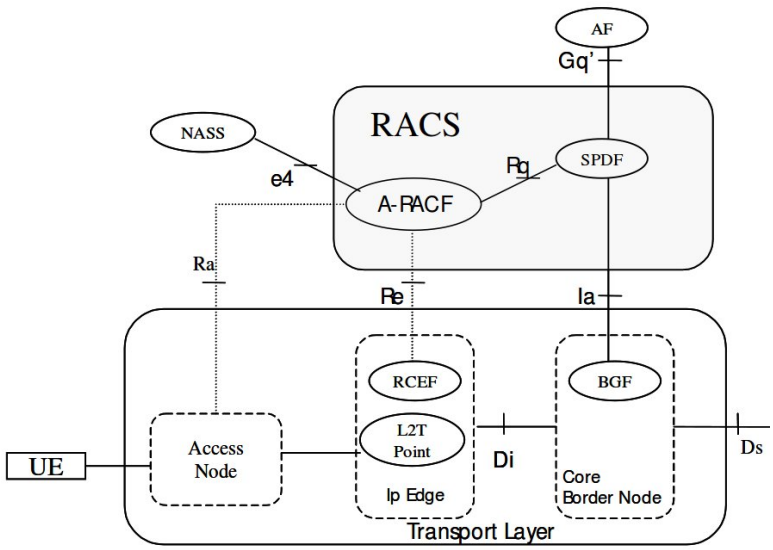


Figure 5.5: Resource and Admission Control Subsystem architecture (taken from (TISPAN, 2006))

- Service Policy Decision Function (SPDF) communicates with the Application Function (AF), a P-CSCF for instance. The AF provides the resource requirements from the session descriptor. The SPDF can take the decision to authorize the QoS resources, which are communicated to the A-RACF or the C-BGF.

- Access Resource and Admission Control Function (A-RACF) is located in the access network and is responsible for directing the SPDF decisions in the RCEF. Depending upon the mode—Guaranteed QoS or diffserv—it sets the appropriate Layer 2/Layer 3 policies or diffserv markers, respectively in the RCEF.

- Resource Control Enforcement Function (RCEF) is located in the access network and can apply the gating function on the Layer 2 termination.

- Border Gateway Function (BGF) supports border control functionality at the IP packet level in the access, core, or inter-core network. The BGF supports the functions for IPv4/IPv6 conversion, NAPT and Network Address Translation (NAT) traversal, traffic screening, and topology hiding. Function Core Border Gateway Function (C-BGF) is an edge router located at the border of the core network. The SPDF can take the decisions to apply a gating function to the IP-flows at the C-BGF.

Figure 5.6 shows the network components associated with QoS control and the control procedure. The control procedure is described as follows.

< 1 > In the access authentication when starting the access to the network from a terminal, RACS receives and retains the QoS profile from NASS.

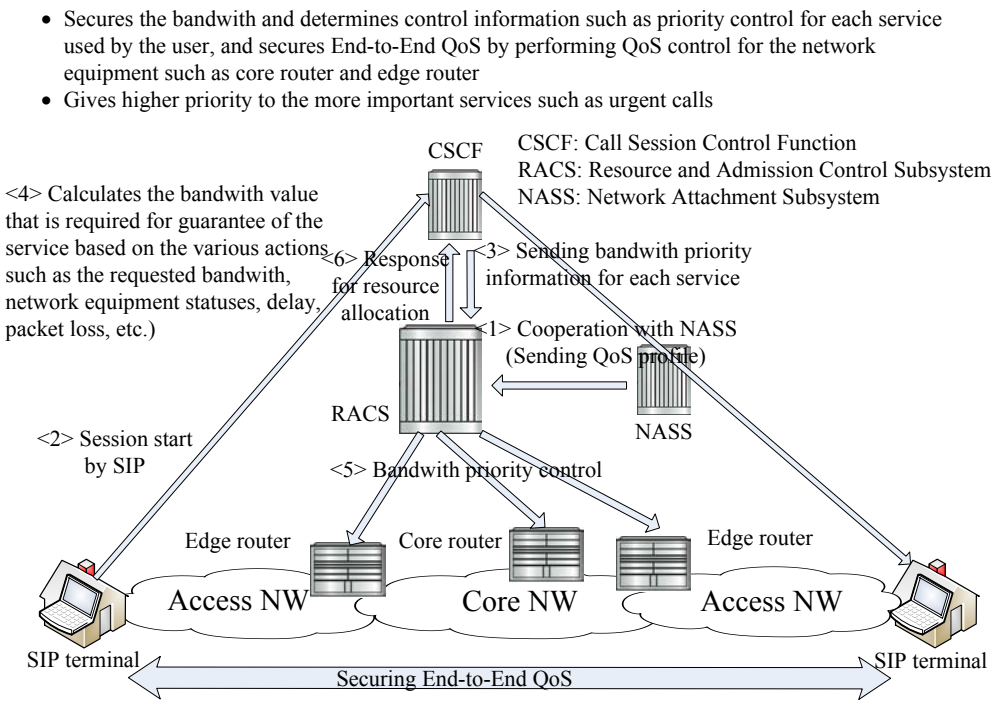


Figure 5.6: Network components and control flow for QoS

< 2 > The originating terminal sends a session initiation request (SIP message) to CSCF.

< 3 > CSCF sends a QoS resource request to RACS together with the information such as the bandwidth required for service execution, service class, and reservation priority.

< 4 > After receiving the QoS resource request, RACS collates the requested conditions with the retained user profile and edge router information, and performs admission control. RACS also calculates the bandwidth value required for guaranteeing the service quality.

< 5 > RACS performs gate control for the edge router based on the calculated bandwidth value.

< 6 > RACS responds about the securing of the QoS resource to CSCF.

< 7 > CSCF sends a session initiation request to the terminating terminal.

5.2.3 NASS subsystem

NASS subsystem (see Figure 5.7) performs IP address distribution and authentication. Network attachment is provided based on either implicit or explicit user identification

credentials stored in its database (respectively, physical or logical Layer 2 addresses, or user name and password).

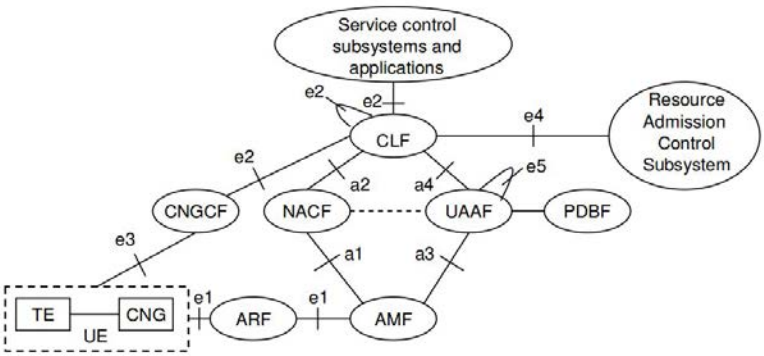


Figure 5.7: Network Attachment Subsystem architecture (taken from (TISPAN, 2010a))

NASS consists of the following functions blocks:

- Network Access Configuration Function (NACF) is responsible for the IP address allocation to the user equipment and it may provide some additional parameters. This service can be provided by a Dynamic Host Configuration Protocol (DHCP) server.
- Access Relay Function (ARF) is a relay between the CNG and the NASS that inserts local configuration information.
- Access Management Function (AMF) translates the network access requests sent by the User Equipment (UE) and forwards requests for allocation of an IP address and possibly additional network configuration parameters to/from the NACF. The AMF also forwards requests to the UAAF to authenticate the user, authorize or deny the network access, and retrieve user-specific access configuration parameters.
- Connectivity Session Location Function (CLF) is used to associate the user IP address to the physical location information and to transfer QoS profile information containing the IP address and QoS information to RACS.
- Profile Database Function (PDBF) stores the user profiles and authentication data, manages user profile information such as subscriber ID and subscribed services.
- User Access Authorization Function (UAAF) performs authentication for network access, based on the user profile stored in the PDBF.
- Customer Network Gateway Configuration Function (CNGCF) is used to configure the Customer Network Gateway (CNG) when necessary.

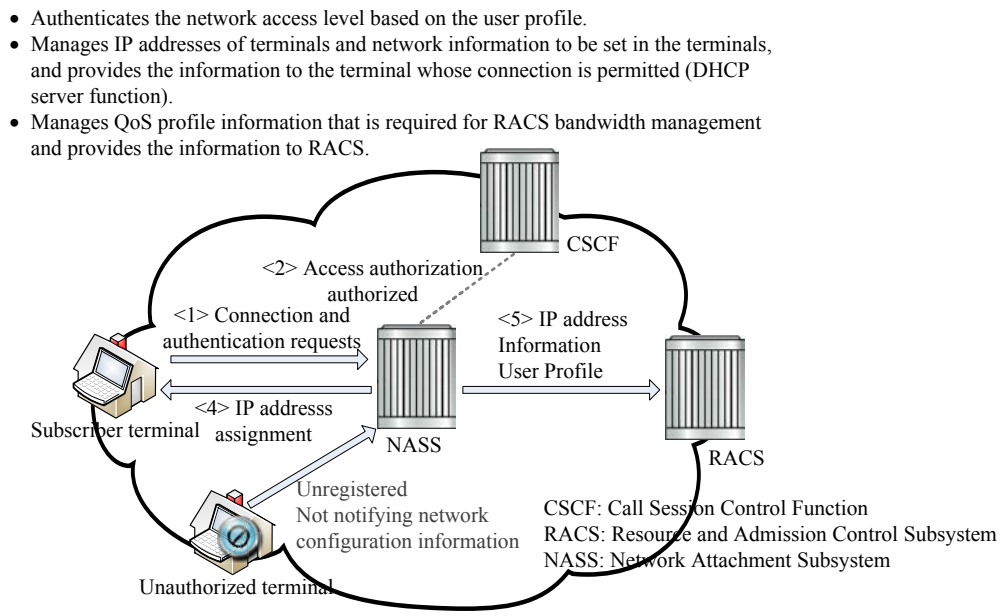


Figure 5.8: Network components and procedure for authentication

Figure 5.8 shows the authentication procedure and related network components. The authentication procedure is described as follows:

< 1 > Terminal requests access authentication to NASS when initiating the access to the network.

< 2 > NASS verifies the profile information such as the subscriber ID, physical access ID, and logical access ID for the terminal connection request, and performs authentication.

< 3 > The authenticated terminal requests IP address assignment to NASS by sending a DHCP request.

< 4 > NASS assigns an IP address to the terminal through DHCP.

< 5 > When interacting with RACS, NASS sends the QoS profile information to RACS together with the subscriber ID and the IP address. (See <1> of the QoS control procedure.

Properties of the functional layer

Figure 5.9 displays the TISPAN IMS functional architecture with NASS, RACS and Core IMS functions connected via interfaces. Such a connection is modelled by a 'connected-to' relationship between a function and an interface. Besides, the functional layer architecture is essentially hierarchical: it is made of subsystems being decomposable into other functions connected via interfaces.

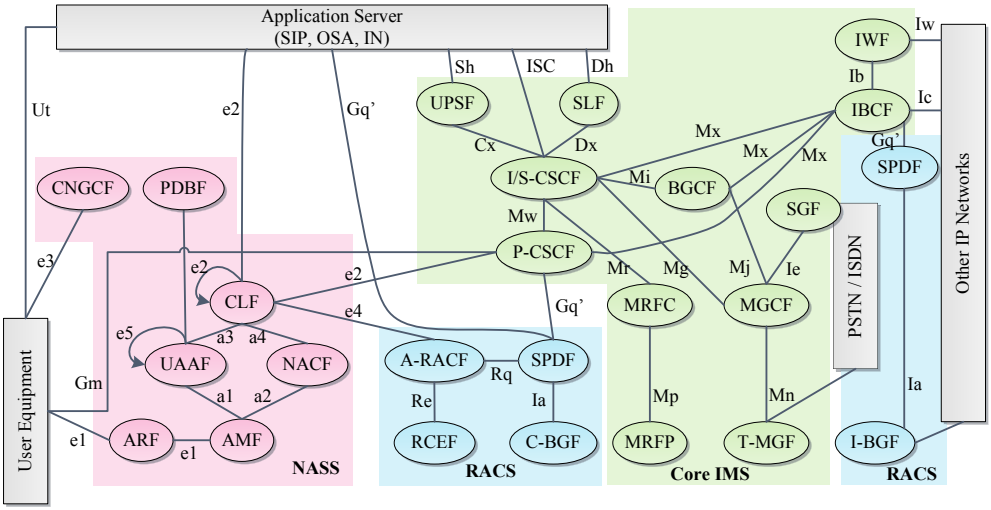


Figure 5.9: TISPAN NGN IMS functional architecture

Mapping between functional and physical layers

In a real network, the IMS functions or logical servers need to be implemented on physical nodes. How to map logical servers located in a functional architecture to physical nodes located in a physical architecture is not standardized, but is of great interest to the network providers. Network providers may choose different mapping strategies to achieve their own objectives. For example, an industry-leading network provider may want a mapping strategy that provides high reliability and high expandability. Moreover, each mapping strategy has its own advantages and disadvantages. Network providers select a mapping strategy with the best performance results according to their needs and actual network conditions, including the number of users, the capacity of physical nodes, the budget plan, and so forth. This requires the providers to consider both advantages and disadvantages of each mapping strategy, in order to determine the one that is satisfied by themselves and their users.

For example, (Xiao et al., 2010) presents a generic mapping strategy and two special mapping strategies. The Generic Mapping Strategy is a method that allows for the mapping of a logical entity into any physical node. In this case, any physical node can host one or more logical server(s). On the other hand, two or more physical nodes can host one or more identical logical servers. Any two or more physical nodes can be identical, which means that they can host the same logical servers. The generic mapping strategy includes all possible mapping ways.

When using the first special mapping strategy (Customized Mapping Strategy 1) presented in (Xiao et al., 2010) to map logical servers to the physical nodes, each physical node only hosts one type of logical server. This mapping strategy is desired when the loads of two or more logical servers exceed the capacity of a physical node. This is often the case for network providers with a large number of users. Overall,

there are three advantages by using this mapping strategy. First, it is easier to create a backup physical node and upgrade capacity for the future. Second, this strategy brings a small impact to the system when the failure occurs on the physical nodes, because each physical node only takes care of one type tasks. Third, it is easy to implement and maintain the physical nodes. The main disadvantages of this mapping strategy is cost. The remaining capacity of the physical nodes which host one type of logical server cannot be allocated to other logical servers and therefore will be wasted.

The second special mapping strategy (Customized Mapping Strategy 2) presented in (Xiao et al., 2010) fits small carriers who try to pack different logical servers into the same physical nodes to save footprint and cost. In this strategy, physical nodes are divided into different groups. One logical server can be hosted by the physical nodes located in one group only. One group of physical nodes can host one or more than one logical servers. Furthermore, it is assumed that a message traversing the logical servers that belong to the same group will be processed by one physical node only in the group. This constraint can reduce the travelling time within a group. In the extreme case, if each group hosts only one logical server, this becomes the customized strategy 1. The main disadvantage of this mapping strategy over the previous one is the complexity running these servers. When a physical node hosts more than one logical server, the physical node has to handle more types of tasks which may interfere with each other. The maintenance cost will clearly be higher.

Figure 5.10 reflects a possible implementation of functions over physical equipment. This embedding is modelled by a 'supported-by' relation between a function and a physical equipment. This is a first example of resource dependency since the failure of an equipment generally impacts the functions running over it.

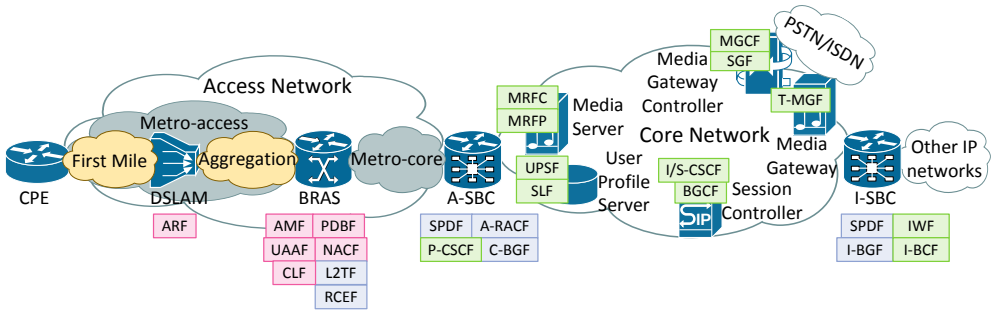


Figure 5.10: Mapping functions to physical equipment

5.3 Procedural layer of the IMS Architecture

Before an IMS terminal starts any IMS-related operation there are a number of prerequisites that have to be met. First, the IMS terminal needs to get access to an IP Connectivity Access Network (IP-CAN) such as Asymmetric Digital Subscriber Line (ADSL). As part of this prerequisite the IMS terminal needs to acquire an IP address.

Second, the IMS terminal needs to discover the IP address of the P-CSCF that will be acting as an outbound/inbound SIP proxy server. When these two prerequisites are fulfilled the IMS terminal registers at the SIP application level to the IMS network. This is accomplished by regular SIP registration. IMS terminals need to register with the IMS before initiating or receiving any other SIP signaling. The IMS registration procedure allows the IMS network to locate the user (i.e. the IMS obtains the terminal's IP address). It also allows the IMS network to authenticate the user, establish security associations, and authorize the establishment of sessions.

The procedures required to meet the above prerequisites are defined in standards. Some procedures involve interactions between NASS functions (e.g. network attachment procedures (TISPAN, 2010a)), others RACS functions (e.g. resource reservation and resource release (TISPAN, 2006)) and others Core IMS functions (e.g. registration (TISPAN, 2010c,b), call origination (TISPAN, 2008a,b).

5.3.1 NASS procedures

The Network attachment procedure (TISPAN, 2010a) is composed of several procedure-stages: authentication and authorization, IP configuration, CNG configuration and Location Management. Depending on the protocols (e.g. Point-to-Point Protocol (PPP), DHCP, etc.) and deployment scenarios used, these procedure-stages can be applied in a different order. Though for security reasons it needs to be ensured that the authentication procedure-stage is always successfully completed first.

Network attachment using DHCP Figure 5.11 shows the network attachment procedure using DHCP with a focus on procedure-stages 1 and 2a of the network attachment process (i.e. authentication and IP address allocation). Note that procedure-stages 2b (P-CSCF IP address discovery), 3 (CNG configuration), and 4 (Location Management) are not considered here.

< 1 > The UE initiates the IP address allocation and implicit authentication procedure by sending a DHCP Discover message.

< 2 > ARF receives the message, adds additional information to the DHCP Discover (e.g. line identification), and forwards the message on to AMF.

< 3 > AMF receives the DHCP Discover and sends an access request to the UAAF to authorize the NASS User associated with the UE which sent the DHCP Discover. The association of NASS User profile and UE is facilitated by the line identification information.

< 4 > UAAF responds with an access accept in case a NASS User profile could successfully be associated with the supplied line identification information.

< 5 > AMF sends the DHCP Discover to NACF, which operates as a DHCP server.

< 6-7 > NACF responds with a DHCP Offer to the UE.

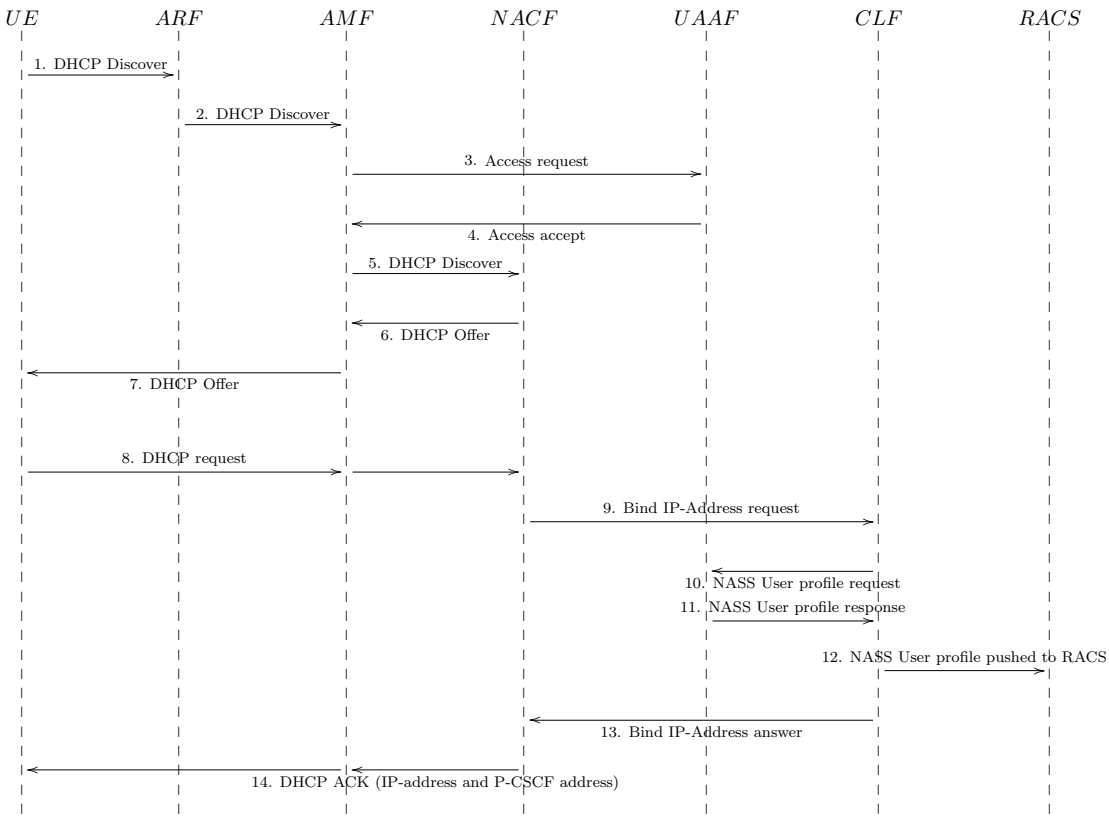


Figure 5.11: Network attachment procedure using DHCP (taken from (TISPAN, 2010a))

< 8 > The UE sends a DHCP Request to request an IP address and through DHCP option 120 the address of a P-CSCF. This request is relayed by the AMF to the NACF.

< 9 > The NACF informs the CLF that an IP address is allocated to the UE.

< 10-11 > The CLF retrieves the NASS User profile from UAAF and associates it with the IP address received.

< 12 > The CLF pushes the NASS User profile along with the associated IP addressing and location information to RACS via the e4 reference point.

< 13 > CLF acknowledges to NACF the successful binding of IP address to NASS User profile. This message may contain address information of the TISPAN NGN Service/Applications Subsystems contact point.

< 14 > NACF provides the allocated IP address as well as the Fully Qualified Domain Name (FQDN) or IP address of the P-CSCF, which is relayed by the AMF to the UE.

5.3.2 RACS procedures

This subsection describes the RACS interactions for notifying the A-RACF when a subscriber attaches to the network and for resource reservation.

Notification to the RACS during network attachment The NASS is responsible for notifying the A-RACF when a subscriber attaches to the network. The NASS provides to A-RACF an association between Subscriber ID/IP address, the bearer used in the access network and additional subscriber access information. Figure 5.12 presents the associated procedure:

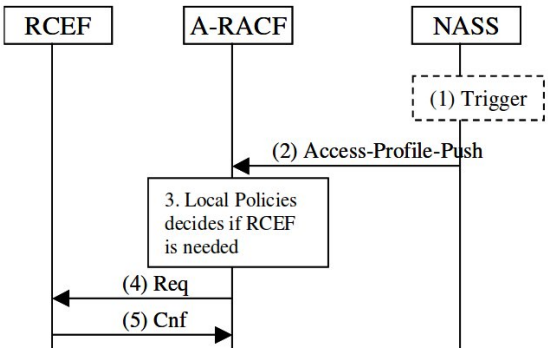


Figure 5.12: Notification sent to the RACS when a user attaches to the network (taken from (TISPAN, 2006))

< 1 > The NASS accepts a request from a customer equipment device to obtain bearer resources to attach to the access network or a modification on a subscriber's access profile that has been previously pushed to the RACS by NASS occurs.

< 2 > The NASS sends Access-Profile-Push to inform A-RACF.

< 3 > Based on Local Policies in the A-RACF and the information received from the NASS, the A-RACF decides if any traffic policy need to be installed, changed or removed. The application of the new local policies will apply to new SPDF requests whereas the current reservations are optionally handled according to previous local policies.

< 4 > The A-RACF requests the RCEF to install traffic policies (depending on step < 3 >).

< 5 > The RCEF confirms the installation of the traffic policies (depending on step < 4 >).

5.3.3 Core IMS procedures

IMS-level registration is accomplished by a SIP REGISTER request. SIP registration is the procedure whereby a user binds his public Uniform Resource Identifier (URI) to a URI that contains the host name or IP address of the terminal where the user is logged in. Registration with the IMS is mandatory before the IMS terminal can establish a session.

IMS registration Figure 5.13 illustrates the registration procedure during which the user is authenticated and authorized to access the IMS network. The goal is achieved and the procedure completes after two round trips. The registration is composed of two procedure-stages. In the first procedure-stage, the network challenges the user while in the second procedure-stage, the user responds to the challenge and completes the registration.

< 1 > The IMS terminal creates a SIP REGISTER request including the user public URI (SIP address) and the IP address of the terminal where the user is logged in. Then, the IMS terminal sends this request to its P-CSCF.

< 2 > The P-CSCF needs to locate an entry point into the user IMS network by executing some DNS procedures. These procedures provide the P-CSCF with the SIP URI of an I-CSCF. The P-CSCF forwards the SIP REGISTER request to this I-CSCF.

< 3 > The I-CSCF does not keep any state associated to registration so it is not aware of whether an S-CSCF is allocated to the user and what the address of such an S-CSCF would be. In order to discover whether there is an S-CSCF already allocated to the user, the I-CSCF sends a Diameter User-Authentication-Request (UAR) to the UPSF.

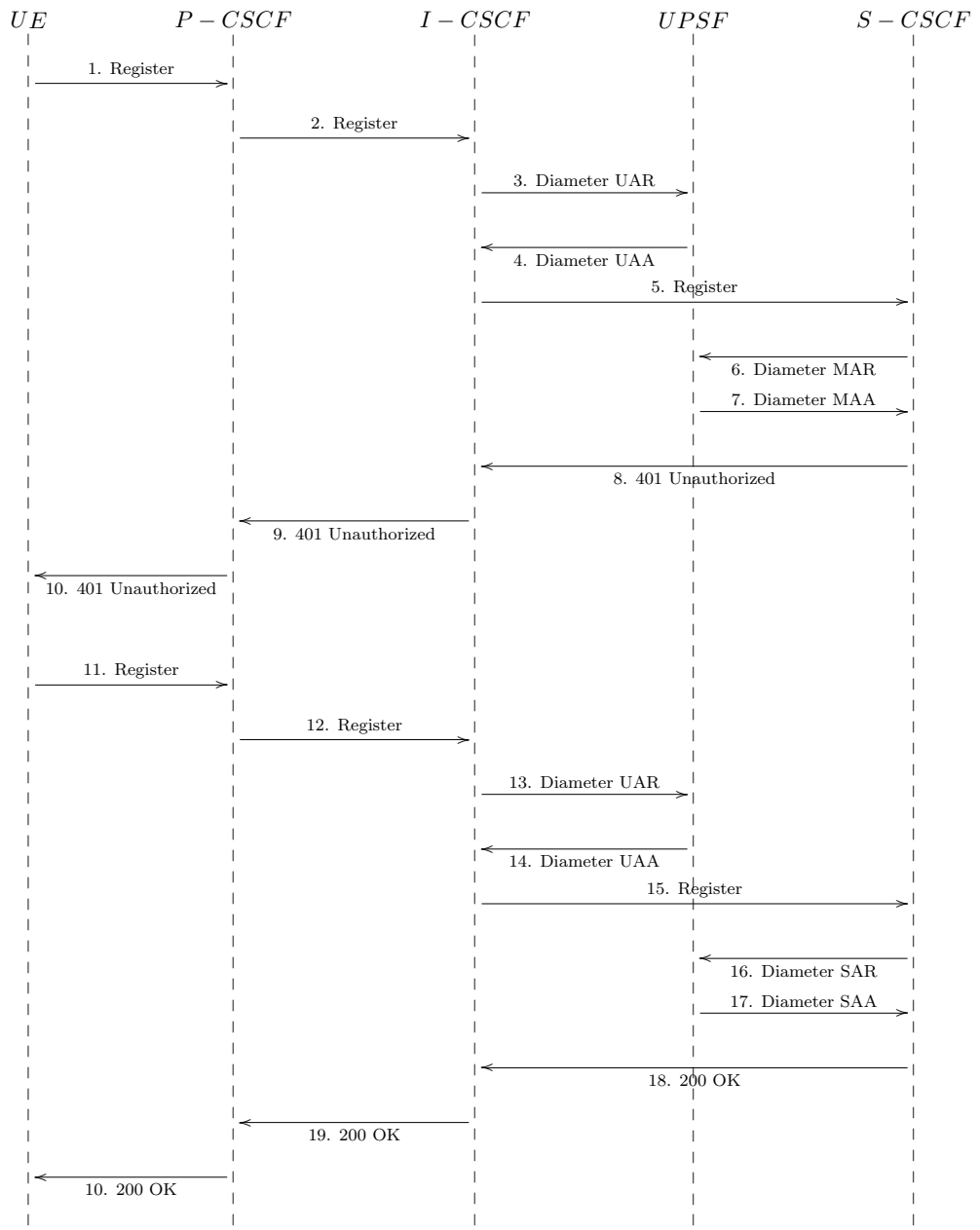


Figure 5.13: IMS-level registration procedure

< 4 > The UPSF answers with a Diameter User-Authentication-Answer (UAA). This answer contains a set of S-CSCF capabilities that are the input for the I-CSCF when selecting the S-CSCF. After performing the S-CSCF selection, the I-CSCF continues with the process by proxying the SIP REGISTER request to the chosen S-CSCF.

< 5 > The S-CSCF receives the REGISTER request and is in charge of authenticating the user.

< 6 > The S-CSCF then contacts the UPSF for a double purpose. On the one hand, the S-CSCF needs to download authentication data to perform authentication for this particular user. On the other hand, the S-CSCF needs to save the S-CSCF URI in the UPSF, so that any further query to the UPSF for the same user will return routing information pointing to this S-CSCF. For this purpose, the S-CSCF creates a Diameter Multimedia-Auth-Request (MAR) message and sends it to the UPSF.

< 7 > The UPSF stores the S-CSCF URI in the user data and answers in a Diameter Multimedia-Auth-Answer (MAA) message. Users are authenticated by the S-CSCF with authentication vectors provided by the UPSF. The UPSF includes one or more authentication vectors in the Diameter MAA message, so that the S-CSCF can properly authenticate the user.

< 8-10 > Then, the S-CSCF creates a SIP 401 Unauthorized response. This response includes a challenge that the IMS terminal should answer. The SIP 401 Unauthorized response is forwarded, according to regular SIP procedures, via the I-CSCF and P-CSCF.

< 11 > When the IMS terminal receives the SIP 401 Unauthorized response, it realizes that there is a challenge included and produces an appropriate response to that challenge. The response to the challenge (sometimes known as credentials) is included in a new SIP REGISTER request sent to the P-CSCF.

< 12 > The P-CSCF does the same operation as for the first REGISTER request; that is, it determines the entry point in the network and finds an I-CSCF.

< 13 > The I-CSCF sends a new Diameter UAR message, for the same reasons as explained before.

< 14 > The difference in this situation is that the Diameter UAA message includes the SIP URI of the S-CSCF allocated to the user. The UPSF stored this URI when it received a Diameter MAR message (6).

< 15 > The S-CSCF receives the REGISTER request that includes the user credentials. It then validates these credentials against the authentication vectors provided by the UPSF in a Diameter MAA message (7).

< 16-17 > If authentication is successful, then the S-CSCF sends a Diameter SAR message to the UPSF to inform the UPSF that the user is now registered and to download the user profile. At this stage, the S-CSCF has stored the IP address for this user, as it was present in the SIP REGISTER request. It has also stored the P-CSCF URI. Later, the S-CSCF will route initial SIP requests addressed to the user via this P-CSCF.

< 18 > The S-CSCF sends a 200 OK response to the REGISTER request, to indicate the success of the REGISTER request. The 200 OK response also contains the address of the S-CSCF of the user. Future SIP requests (excluding REGISTER requests, which are always routed according to the instructions received from the UPSF) that the IMS terminal sends will be routed via this S-CSCF, in addition to the outbound proxy (P-CSCF).

< 19-20 > The 200 OK response traverses the same I-CSCF and P-CSCF that the REGISTER request traversed. Eventually, the IMS terminal gets the 200 OK response. At this stage the registration procedure is complete.

IMS call origination Figure 5.14 shows the origination procedure for a UE located in the home network.

< 1 > UE sends the SIP INVITE request, containing an initial SDP, to the P-CSCF address determined with P-CSCF discovery mechanism. The initial SDP may represent one or more media for a multimedia session.

< 2 > A connection is reserved in the C-BGF with optional NAT binding list retrieval.

< 3 > P-CSCF remembers (from the registration procedure) the next hop CSCF for this UE. In this case it forwards the INVITE to the S-CSCF in the home network.

< 4 > S-CSCF validates the service profile, and invokes any origination service logic required for this user. This includes authorisation of the requested SDP based on the user's subscription for multimedia services.

< 5 > S-CSCF forwards the request, as specified by the S-S procedures.

< 6 > The media stream capabilities of the destination are returned along the signalling path, per the S-S procedures.

< 7-9 > S-CSCF forwards the Offer Response message to the P-CSCF which triggers RACS. RACS performs admission control based on the Offer and Answer parameters. RACS configures the connections in the C-BGF based on the SDP answer and optionally requests a NAT binding list.

< 10 > UE decides the offered set of media streams for this session, confirms receipt of the Offer Response and sends the Response Confirmation to P-CSCF. The Response Confirmation may also contain SDP. This may be the same SDP as in

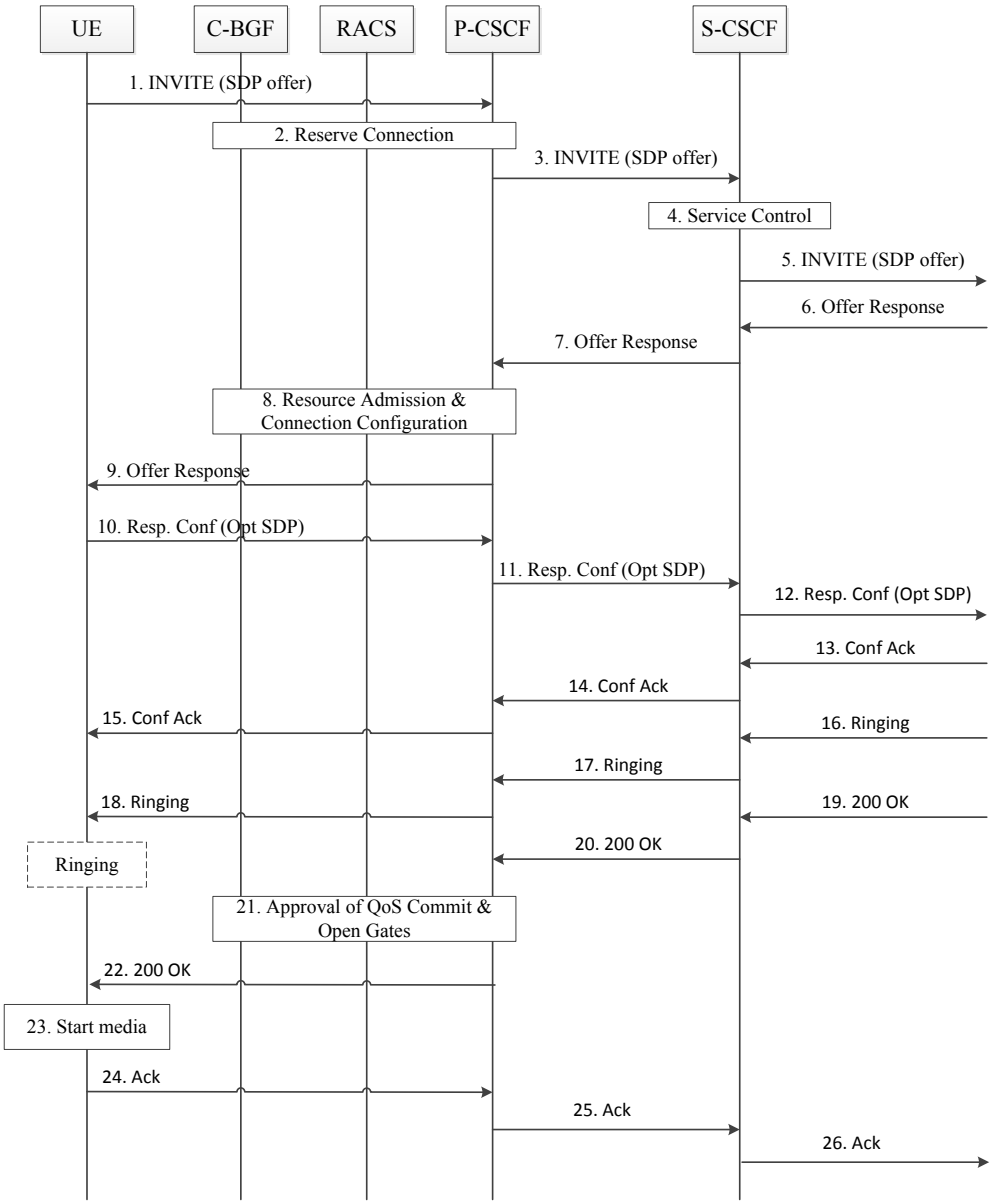


Figure 5.14: Call origination procedure (taken from (TISPAN, 2008a))

the Offer Response received in Step 9 or a subset. If new media are defined by this SDP, a new connection configuration shall be performed following Step 2. The originating UE is free to continue to offer new media in this request or in subsequent requests using the Update method. Each offer/answer exchange will cause the P-CSCF to repeat the RACS interactions again.

< 11 > P-CSCF forwards this message to S-CSCF.

< 12 > S-CSCF forwards this message to the terminating endpoint, as per the S-S procedure.

< 13 > The terminating end point responds to the originating end with an acknowledgement. If Optional SDP is contained in the Response Confirmation, the Confirmation Acknowledge will also contain an SDP response. If the SDP has changed, the admission control and configure connection flows are repeated.

< 14-15 > S-CSCF and P-CSCF forward the answered media towards the UE.

< 16-18 > The destination UE may optionally perform alerting. If so, it signals this to the originating party by a provisional response indicating Ringing. This message is sent to S-CSCF per the S-S procedure. It is sent from there toward the originating end along the signalling path. UE indicates to the originating user that the destination is ringing.

< 19-20 > When the destination party answers, the terminating endpoint sends a SIP 200-OK final response along the signalling path to the originating endpoint, as specified by the termination procedures and the S-S procedures.

< 21 > P-CSCF performs the approval of QoS Commit procedure which triggers the Open Gates procedures if required.

< 22 > P-CSCF passes the 200-OK response back to UE

< 23 > UE starts the media flow(s) for this session.

< 24-26 > UE responds to the 200 OK with an ACK message which is sent to P-CSCF and passed along the signaling path to the terminating endpoint.

Properties of the procedural layer

The procedures above have a multi-resolution structure: they decompose into stages or phases, that themselves decompose into sequences (or partial orders) of request/response between functional elements of the NASS, RACS and Core IMS. Each such request/answer decomposes further into exchanges over interfaces that obey standard protocols, such as DHCP, SIP or Diameter. The execution of a procedure positions state variables, some of which are observable or can be tested. For example, the correct obtention of an IP address proves that the network attachment of the User Equipment (via the NASS) was correctly performed. Procedures, phases, and their smaller elements depend on each

other, in the sense that they are (partially) ordered in time, which we denote by an 'executed before' relation. But they are also 'supported by' the functions and interfaces of the functional layer that run these procedures. This information is easily accessible in the standards, and reveals another form of dependency between resources.

Our approach has focused essentially on the signalling part of an IMS network, as much information is available in the standards and can be used for diagnosing malfunctions. The transport plane could be partly addressed in that manner, but less information is available on the different problems one may encounter, on their causes and their consequences. For example, scarse (expert) knowledge exists relating the use of an incompatible vocoder and the perceived quality on the user side. But one may be able to discover that a RACS policy prevents one of the end users to make hifi VoIP calls.

5.4 Contributions: Relevant structural properties

Figure 5.15 illustrates the generic architecture of an IMS network, which we organized into three layers, each one gathering resources of a specific nature. This architecture exhibits some structural properties that are relevant and general enough to be taken into account in the design of the dependency model that will serve as basis for diagnosis operations.

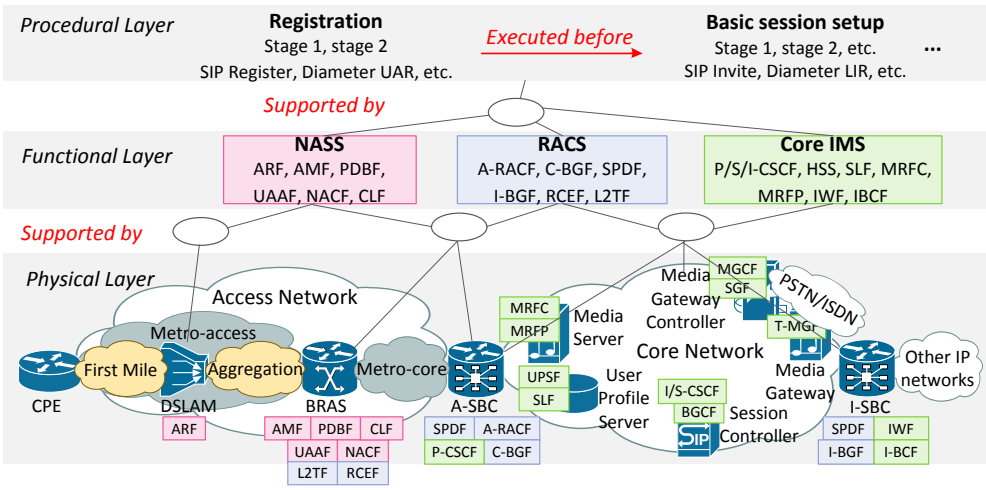


Figure 5.15: IMS networks are hierarchically organized structures

The physical layer architecture is made of specific equipment connecting various network segments. We represent such a connection by a 'connected-to' relationship between an equipment and a segment. Besides, the physical layer architecture is hierarchical, in the sense that network segments are decomposable into other physical equipment connecting smaller network segments.

The functional layer architecture is made of functions connected via interfaces. We represent such a connection by a 'connected-to' relationship between a function and an interface. Besides, the functional layer architecture is also hierarchical, in the sense that it is made of subsystems being decomposable into other functions connected via interfaces.

We represent the fact that a function is implemented into a physical equipment by a 'supported-by' relation between the function and the equipment. This is a first example of resource dependency since the failure of an equipment generally impacts the functions running over it.

The procedures in the procedural layer have a multi-resolution structure: they decompose into stages or phases, that themselves decompose into sequences (or partial orders) of request/response between functional elements.

Each such request/response describes exchanges or interactions over interfaces that obey standard protocols. Procedures, phases, and their smaller elements depend on each other, in the sense that they are (partially) ordered in time, which we denote by an 'executed before' relation.

Furthermore, procedures and phases are also 'supported by' the functions and interfaces of the functional layer that run these procedures.

Chapter 6

Network model describing resource dependencies

Contents

6.1 Different types of resources and their relationships 113

6.1.1 Physical resources and their relationships 114

6.1.2 Functional resources and their relationships 115

6.1.3 Procedural resources and their dependencies 117

6.1.4 Inter-layer relationships 118

6.2 Contributions: Models of network dependencies 120

6.2.1 Network model class hierarchy 120

6.2.2 Causal graph of network dependencies 121

6.2.3 Generic model vs network instance: self-modeling 123

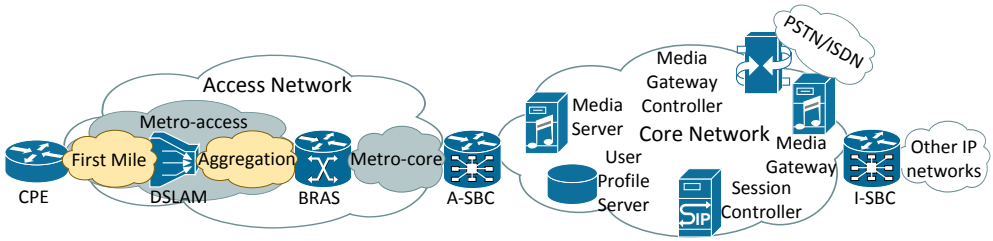
The three-layer model described in the previous chapter is *generic*, in the sense that it defines the different *types* of network resources involved (for a given range of technologies), how they depend on each other and how they interact. The generic model defines the building blocks of a network, from a management perspective, but the actual network one has to manage contains many *instances* of these elements. The actual network can be represented as a large collection of instances of the patterns described in the generic model, and these instances overlap on some common resources. In this chapter, we use object oriented paradigm to represent IMS network resources and their relationships.

6.1 Different types of resources and their relationships

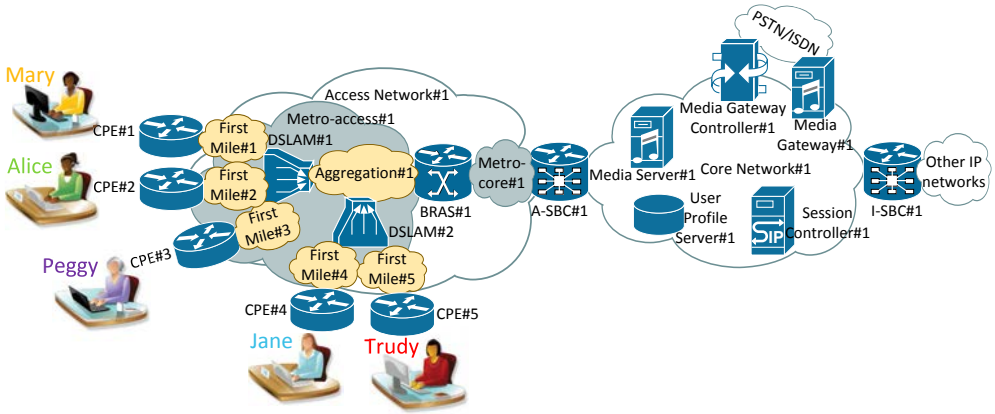
Our representation of IMS resources and dependencies is based on six important object classes: *Equipment*, *Link*, *Function*, *Interface*, *Procedure*, and *Message* such that each IMS resource inherits from (i.e. is a subclass of) one of the above classes; and four important relationships: *is-connected-to*, *is-composed-of*, *is-supported-by*, *is-preceded-by*.

6.1.1 Physical resources and their relationships

The physical layer contains *physical resources* that belongs to two classes/types: equipment or link. We use Figure 6.1(a) and Figure 6.1(b) to describe these resources and illustrate the relationships between them.



(a) The generic IMS physical architecture



(b) An instance of the generic IMS physical architecture

Figure 6.1: The generic IMS physical architecture and its instance

- ***Equipment***. An instance of the *Equipment* class within the physical layer represents a physical box. For example, CPE, displayed in Figure 6.1(a), is one of the subclasses inheriting from *Equipment*. Other subclasses include DSLAM, BRAS, SBC, etc. In Figure 6.1(b), five instances CPE#1, CPE#2, CPE#3, CPE#4, CPE#5 of CPE have been displayed.

- ***Link***. An instance of the *Link* class within the physical layer represents a network segment. For instance, First mile, displayed in Figure 6.1(a), is one of the subclasses inheriting from *Link*. Other subclasses include Aggregation, Metro-access, Metro-core, Access-network, Core-network, etc. In Figure 6.1(b), five instances: first-mile#1, first-mile#2, first-mile#3, first-mile#4, first-mile#5 of this subclass have been displayed.

- ***is-connected-to***. This relationship denotes the fact that an equipment (resp. link) may connect to more than one link (resp. equipment). For example, DSLAM

is connected to first-mile in Figure 6.1(a), while in Figure 6.1(b) DSLAM#1 is connected first-mile#3.

- ***is-composed-of***. This relationship denotes the fact that a link (or an equipment) is composed of other links and equipment. If a physical resource X is composed of several physical resources $X_1, X_2, \cdots, X_n$, the set $\{X_1, X_2, \cdots, X_n\}$ is called the set of *descendants* of X which is said to be the *ancestor* of each X_i, $i = 1 \cdots n$. For example, in Figure 6.1(a), Metro-access is composed of first-mile, DSLAM and Aggregation. As for Figure 6.1(b), it shows that Metro-access#1 is composed of five instances of first-mile (first-mile#1, first-mile#2, first-mile#3, first-mile#4, first-mile#5), two instances of DSLAM (DSLAM#1 and DSLAM#2) and one instance of Aggregation (Aggregation#1). This relationship enables to organize/analyse/view the physical IMS architecture on different levels of granularity. By a refinement operation we go from a coarse view of the architecture to a finest (more detailed) one.

- ***is-connected-to*** and ***is-composed-of***. When (a) a physical resource X is connected to another physical resource Y and (b) X has a non-empty set of descendants $X_1, X_2, \cdots, X_n$, then at least one descendant of X is connected to Y. For example, in Figure 6.1(a) where Metro-access is connected to BRAS and composed of first-mile, DSLAM, and Aggregation, the latter descendant is also connected to BRAS. Similarly, in Figure 6.1(b) Metro-access#1 and its descendant Aggregation#1 are connected to BRAS#1. The connectivity between equipment and links is preserved by the refinement and abstraction operations.

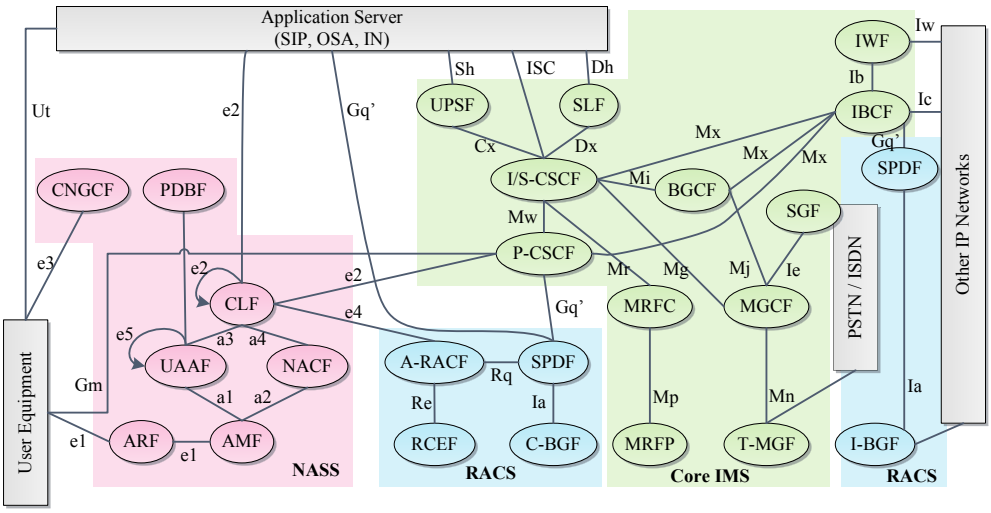
6.1.2 Functional resources and their relationships

The functional layer contains functional resources that are either functions or interfaces. We use Figure 6.1(a) and Figure 6.1(b) to describe these resources and illustrate the relationships between them.

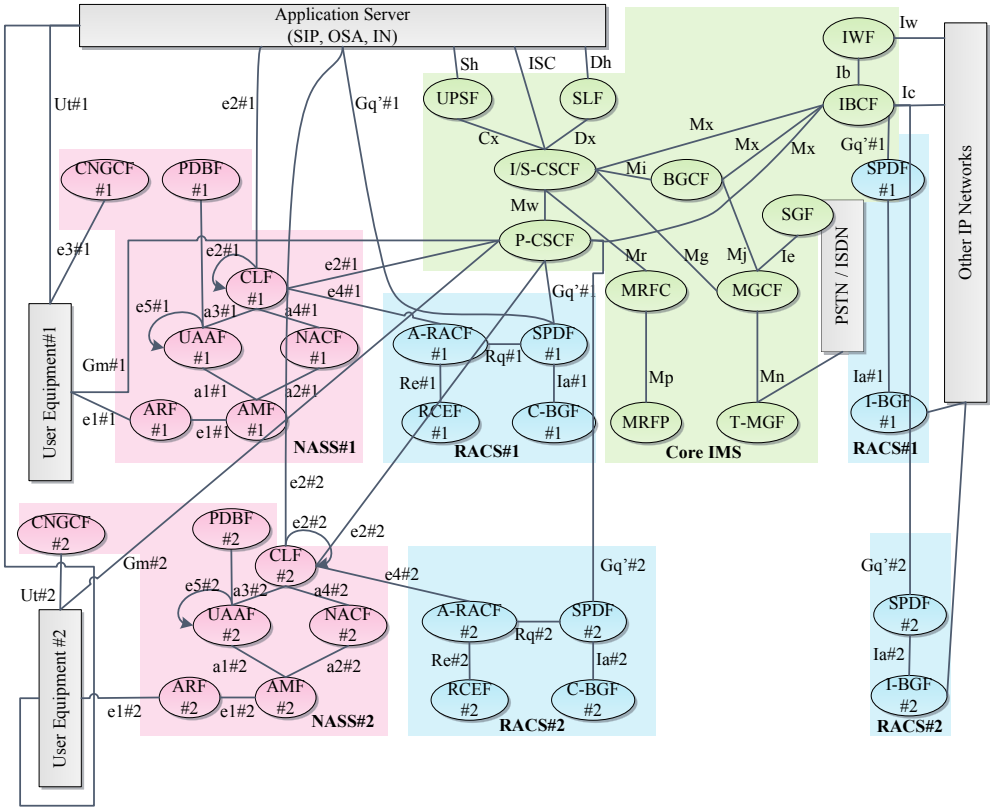
- ***Function***. An instance of the *Function* class within the functional layer represents a logical entity. For example, ARF displayed in Figure 6.2(a) is one of the subclasses inheriting from *Function*. Other subclasses include AMF, NACF, NASS, A-RACF, RACS, P-CSCF, Core-IMS, etc. In Figure 6.2(b), two instances ARF#1 and ARF#2 of ARF have been displayed.

- ***Interface***. An instance of the *Interface* class within the functional layer represents a reference point whereby functions can interact. For instance, e1 (between UE and ARF) displayed in Figure 6.2(a) is one of the subclasses inheriting *Interface*. Other subclasses include Gm (between UE and P-CSCF), e2 (between CLF and Core-IMS), Gq' (between Core-IMS and RACS), etc. In Figure 6.2(b) two instances of e1 (between UE and ARF) have been displayed: e1#1 and e1#2.

- ***is-connected-to***. This relationship denotes the fact that a function (interface) may connect to more than one interface (function). For example, in Figure 6.2(a),



(a) The generic IMS functional architecture



(b) An instance of the generic IMS functional architecture

Figure 6.2: The generic IMS functional architecture and its instance

UE is connected to e1 (between UE and ARF) while in Figure 6.2(b), UE#2 is connected to e1#2 (between UE#2 and ARF#2).

- ***is-composed-of***. This relationship denotes the fact that a function (interface) is composed of other functions and interfaces. When a functional resource X is composed of several functional resources $X_1, X_2, \cdots, X_n$. The set $\{X_1, X_2, \cdots, X_n\}$ is called the set of *descendants* of A which is then said to be the *ancestor* of each A_i, $i = 1 \cdots n$. For example, in Figure 6.2(a) NASS is composed of CNGCF, ARF, AMF, PDBF, UAAF, NACF, and CLF. As for Figure 6.2(b), it shows that NASS#2 is composed of CNGCF#2, ARF#2, AMF#2, PDBF#2, UAAF#2, NACF#2, and CLF#2. This relationship enables to organize/analyse/view the functional IMS architecture on different levels of granularity. By a refinement operation we go from a coarse view of the architecture to a finest (more detailed) one.

- ***is-connected-to*** and ***is-composed-of***. When (i) a functional resource X is connected to another functional resource Y and (ii) X has a non-empty set of descendants $X_1, X_2, \cdots, X_n$, then at least one descendant of X is connected to Y. For example, in Figure 6.2(a) where NASS is connected to A-RACF and is composed of CNGCF, ARF, AMF, PDBF, UAAF, NACF, and CLF. The latter descendant is connected to A-RACF. Similarly in Figure 6.2(b), NASS#1 and its descendant CLF#1 are connected to A-RACF#1. The connectivity between functions and interfaces is preserved by the refinement and abstraction operations.

6.1.3 Procedural resources and their dependencies

The procedural layer contains resources that are either procedures or messages. Below we present these resources and the relationships between them.

- ***Procedure***. An instance of the *Procedure* class within the procedural layer represents a procedure, a procedure-stage defined either by the standards or a more complex high-level service capability defined by the human expert. Examples of subclasses inheriting from *Procedure* are network attachment, IMS registration, IMS call origination, etc.

- ***Message***. An instance of the *Message* class within the procedural layer represents a request or a response sent from a function to another one via a reference point. Examples of subclasses inheriting from *Message* are DHCP Discover, DHCP Offer, Register, Diameter UAR, Invite, 200 OK, etc. Sometimes, receiving a specific message or not receiving a message in time can be an explanation for an observed failure.

- ***is-preceded-by***. This relationship denotes the fact that the execution of a procedure (message) requires to have, previously, successfully executed some prerequisites: other procedures or messages. For instance, the IMS registration procedure

is required before a terminal can execute the IMS call origination procedure. Another example is the network attachment procedure, where CNG configuration is preceded by IP configuration. In the same manner, DHCP Offer is preceded by DHCP Discover. When a procedural resource Y is preceded by a procedural resource X, we say that X is a *predecessor* of Y that is then said to be a *successor* of X.

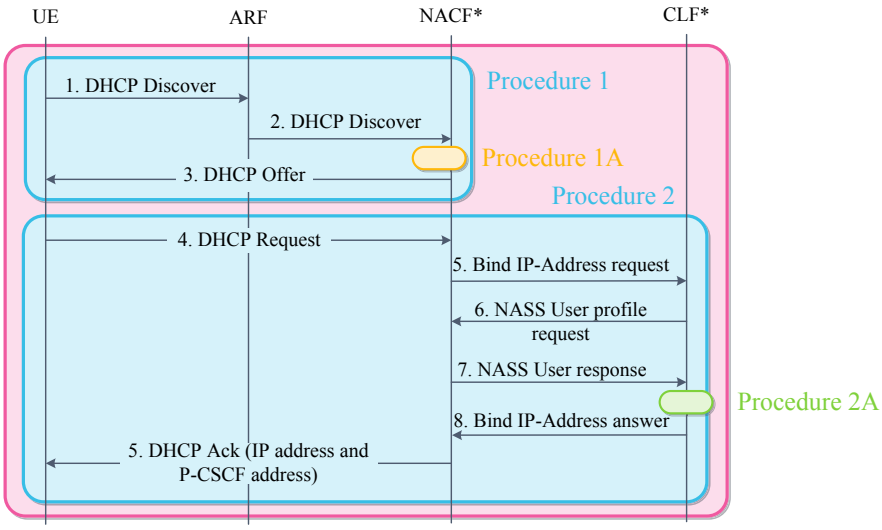
- ***is-composed-of***. This relationship denotes the fact that a procedure is composed of other procedures and messages. Suppose that a functional resource X is composed of the following functional resources $X_1, X_2, \cdots, X_n$. The set $\{X_1, X_2, \cdots, X_n\}$ is the set of *descendants* of X which is then said to be the *ancestor* of each A_i, $i = 1 \cdots n$. For example, the Network attachment procedure as composed of three ordered procedures: IP configuration, CNG configuration and Location Management (TISPAN, 2010a) where IP configuration, displayed in Figure 6.3(a), is composed of *procedure 1* and *procedure 2*. As illustrated in Figure 6.3, this relationship enables to organize/analyse/view a procedure on different levels of granularity. By a refinement operation we go from a coarse view of the procedure to a finest (more detailed) one.

- ***is-preceded-by*** and ***is-composed-of***. When (i) a procedural resource Y is preceded by another procedural resource X and (ii) Y has a non-empty set of descendants $Y_1, Y_2, \cdots, Y_n$, then each descendant of Y is preceded by X. For example, within `IP configuration` procedure displayed in Figure 6.3(b), *procedure 2* (composed of message (8) ... messages (14)) is preceded by *procedure 1* (composed of message (1) ... message (7)). Each descendant of *procedure 2* is preceded by *procedure 1*.

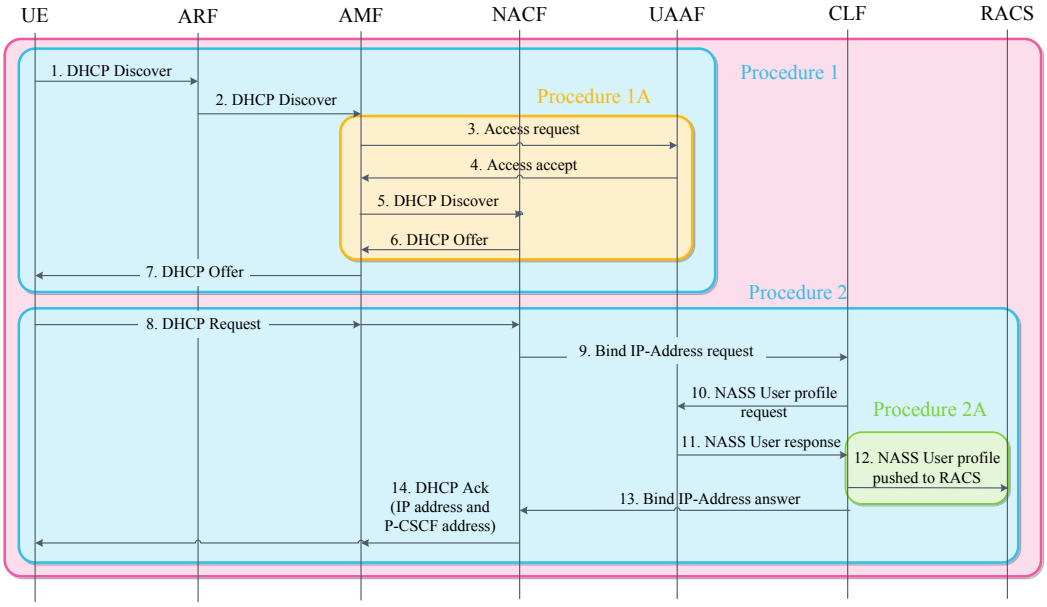
6.1.4 Inter-layer relationships

The mapping between the generic physical and functional architectures of an IMS network describes how *functions* and *interfaces* are implemented into *equipment* and *links*. Figure 6.4 displays an example of mapping. Note that interfaces have not been displayed.

- ***is-supported-by***.
  - ***is-supported-by*** from the physical layer to the functional one. This relationship denotes the fact that a functional resource (a function or an interface) is hosted into a physical resource (an equipment or a link). For example, in Figure 6.4, Core-IMS is supported by Core-network, NACF is supported by BRAS, e1 (between ARF and AMF) is supported by Aggregation.
  - ***is-supported-by*** from the functional layer to the procedural one. This relationship denotes the fact that the execution of a procedural resource (procedure or message) requires interactions of functions via interfaces connecting



(a) Coarse granularity of the IP configuration procedure



(b) Finest granularity of the IP configuration procedure

Figure 6.3: Two different levels of granularity for IP configuration procedure. The coarse granularity distinguishes a single resource denoted 'NACF*' that groups the 'AMF', the 'NACF', the 'UAAF', the communication interface between 'AMF' and 'NACF' ('int AMF-NACF') and the one between 'NACF' and 'UAAF' ('int NACF-UAAF'). In the same manner, the chosen granularity distinguishes a single resource denoted 'CLF*' that groups the 'CLF', the 'A-RACF' and the communication interface ('int CLF-A-RACF') between these two functions.

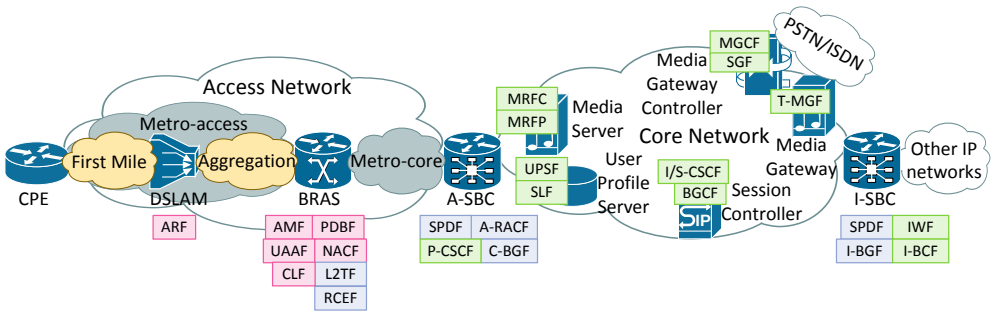


Figure 6.4: A mapping between the physical and the functional architecture (interfaces have not been displayed)

them. As an example, consider Figure 6.3(b) where the IP configuration procedure is supported by NACF and message DHCP discover (2) is supported by e1 (between ARF and AMF).

- ***is-supported-by*** and ***is-composed-of***.
  - When (i) a functional (procedural) resource Y is supported by a physical (functional) resource X and (ii) Y has a non-empty set of descendants $Y_1, Y_2, \cdots, Y_n$, Then, at least one descendant of Y is supported by X. For example, in Figure 6.4 where Core-IMS is supported by A-SBC, one can see that P-CSCF is supported by A-SBC. Similarly, in Figure 6.3(b) where IP configuration procedure is supported by CLF, one can see that *procedure 2* (started with DHCP request (8) and ended with DHCP ack (14)) is supported by CLF.
  - When (i) a functional (procedural) resource Y is supported by a physical (functional) resource X and (ii) X has a non-empty set of descendants $X_1, X_2, \cdots, X_n$, then Y is supported by at least one descendant of X. For instance, in Figure 6.4 where NASS is supported by Metro-network, one can see that NASS is supported by DSLAM. Similarly, the IP configuration procedure displayed in Figure 6.3(a) and supported NACF*, is also supported by NACF (see Figure 6.3(b)).

6.2 Contributions: Models of network dependencies

We used object-oriented paradigm to represent IMS network resources and their relationships. Our model class hierarchy is shown in the class diagram of Figure 6.5.

6.2.1 Network model class hierarchy

Any *Network resource*, no matter the layer to which it belongs, may be broken down into more than one resource within the same layer. This breakdown is expressed by

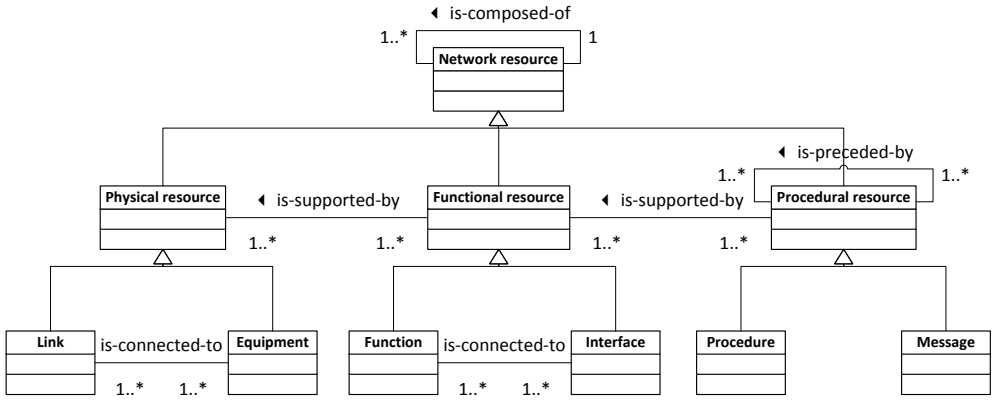


Figure 6.5: Network model class hierarchy

the reflexive association *is-composed-of*.

Each network resource belongs to exactly one specific layer: it may be a *physical resource*, a *functional resource* or a *procedural resource*. A procedural resource has the particular feature that they may be preceded by more than one procedural resource. This precedence is expressed by the reflexive association *is-preceded-by*.

The execution of a procedural resource may be enabled by more than one functional resource. This support provided by the functional resources is expressed by the association *is-supported by* from the procedural resource to the functional one. Similarly, this type of association exists from a functional resource to a physical one expressing the fact that a functional resource may be implemented into more than one physical resource.

A physical resource is either an *Equipment* or a *Link*. A functional resource is either a *Function* or an *Interface*. As for a procedural resource, it is either a *Procedure* or a *Message*. Furthermore, a link (equipment) may connect to more than one equipment (link). In the same manner, a function (interface) may connect to more than one interface (function). This connectivity property is expressed by the association *is-connected-to*.

The above object classes: *Equipment*, *Link*, *Function*, *Interface*, *Procedure*, and *Message* are important since managed IMS resources (e.g. `DSLAM`, `Metro-access`, `NASS`, `P-CSCF`, `network attachment`, `DHCP Discover`, etc) can inherit from one of them. For instance, `UE` inherits from *Function*. Figure 6.6 shows this subclass with two attributes `IP address` and `P-CSCF address`, and a method `IPconnectivity()` that updates these attributes during the network attachment procedure.

6.2.2 Causal graph of network dependencies

The information contained in the model class hierarchy can be expressed by the graph $G = (V, E_{\leftrightarrow}, E_{>}, E_{\Uparrow}, E_{\leftharpoonup})$ described below.

- The set V of vertices denotes the set of IMS network resources. V is partitioned

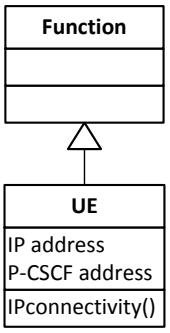


Figure 6.6: Class UE inherits from class Function

into three subsets V_1, V_2, and V_3 where V_1 is the set of physical resources (equipment and links), V_2 the set of functional resources (functions and interfaces), and V_3 the set of procedural resources (procedures and messages).

- The set $E_{\leftrightarrow} \subseteq (V_1 \times V_1) \cup (V_2 \times V_2)$ of edges represents the relationship *is-connected-to*. $\forall (u, v) \in (V_1 \times V_1) \cup (V_2 \times V_2)$, u *is-connected-to* v is written as $u \leftrightarrow v$.

- The set $E_{>} \subseteq (V_1 \times V_1) \cup (V_2 \times V_2) \cup (V_3 \times V_3)$ of arcs represent the relationship *is-composed-of*. $\forall (u, v) \in (V_1 \times V_1) \cup (V_2 \times V_2) \cup (V_3 \times V_3)$, u *is-composed-of* v is written as $u > v$. The set $E_{>}$ defines three directed acyclic graphs: one on V_1, one on V_2 and another one on V_3. For all $u \in V$, we denote $\epsilon(u) = \{u' \in V, u > u'\}$ the set of descendants of u.

- The set $E_{\Uparrow} \subseteq (V_1 \times V_2) \cup (V_2 \times V_3)$ of arcs represents the relationship *is-supported-by*. $\forall (u, v) \in (V_1 \times V_2) \cup (V_2 \times V_3)$, u *is-supported-by* v is written $u \Uparrow v$. The set $E_{\Uparrow}$ defines a directed acyclic graph (DAG) on V.

- The set $E_{\prec} \subseteq (V_3 \times V_3)$ of arcs represents the relationship *is-preceded-by*. $\forall (u, v) \in (V_3 \times V_3)$, u *is-preceded-by* v is written as $u \prec v$. The set $E_{\prec}$ defines a directed acyclic graph (DAG) on V_3.

- Consistency between the relationship *is-composed-of* (refinement operation) and the three other relationships *is-connected-to*, *is-supported-by* and *is-preceded-by*:
  $\forall (u, v) \in (V_1 \times V_1) \cup (V_2 \times V_2), [u \leftrightarrow v \wedge \epsilon(u) \neq \emptyset \leftrightarrow \exists\, u' \in \epsilon(u) : u' \leftrightarrow v \wedge v \notin \epsilon(u)]$
  $\forall (u, v) \in (V_1 \times V_2) \cup (V_2 \times V_3), [u \Uparrow v \wedge \epsilon(v) \neq \emptyset \leftrightarrow \exists\, v' \in \epsilon(v) : u \Uparrow v']$
  $\forall (u, v) \in (V_1 \times V_2) \cup (V_2 \times V_3), [u \Uparrow v \wedge \epsilon(u) \neq \emptyset \leftrightarrow \exists\, u' \in \epsilon(u) : u' \Uparrow v]$
  $\forall (u, v) \in (V_3 \times V_3), [u \prec v \wedge \epsilon(v) \neq \emptyset \leftrightarrow \forall v' \in \epsilon(v),\, u \prec v']$
  $\forall (u, v) \in (V_3 \times V_3), [u \prec v \wedge \epsilon(u) \neq \emptyset \leftrightarrow \forall u' \in \epsilon(u),\, u' \prec v]$

The relationships *is-supported-by* and *is-preceded-by* are used to express causal relations between failure events occurring in the IMS network. From graph G, above

a causal graph $H = (V, E_{\uparrow})$ where $E_{\uparrow} = E_{\Uparrow} \cup E_{\leftarrow}$ can be constructed. In graph H, it suffices to define the strength of the causal links, represented as conditional probabilities, in order to obtain a Bayesian network for reasoning under uncertainty. The relationship *is-connected-to* serves to identify a specific network resource instance. As for the relationship *is-composed-of*, it can be useful to obtain a hierarchical representation of the causal graph, each level corresponds to a particular level of abstraction for the network resources. Nevertheless, as a first step we ignore the refinement operation and choose to reason with a single level of granularity.

6.2.3 Generic model vs network instance: self-modeling

The three-layer model described in the previous chapter is *generic*, in the sense that it defines the different *types* of network resources involved (for a given range of technologies), how they depend on each other and how they interact. Most of it derives from the standards or best practices of an operator, but part of it must be designed by an expert. This work is facilitated by the relatively small size of this generic model, by the existence of a grammar defining how objects can be related, and by the hierarchical nature of the model. This allows one to model each layer by progressive refinements, thus selecting the finest granularity at which one wishes to manage the network.

The generic model defines the building blocks of a network, from a management perspective, but the actual network one has to manage contains many *instances* of these elements. Borrowing to object oriented programming, the generic model is a class diagram, while one has to monitor an object diagram, with many objects of the same class. The actual network can be represented as a large collection of instances of the patterns described in the generic model, and these instances overlap on some common resources. This is illustrated in Figure 6.7, where several users of an access network have private parts (CPE and first mile), but may or not share a DSLAM, may or not be connected to the same aggregation segment, etc.

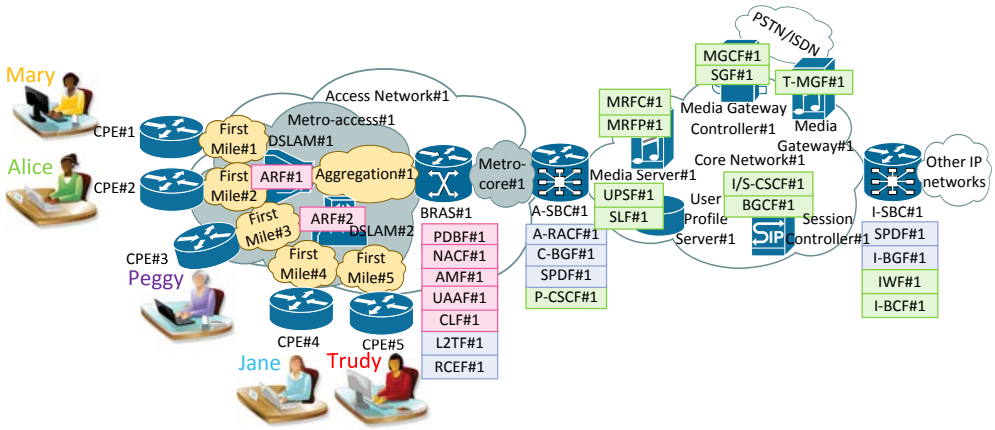


Figure 6.7: An IMS network instance: physical layer, with mention of the supported functions of the functional layer.

Deriving the model of the actual network instance to be managed means creating as many instances as necessary of the network resources (equipment, functions, etc), as they are described in the generic model. It also means structuring or connecting them in accordance with the patterns allowed by the generic model. Such a construction guarantees both the adequacy of the resulting model with the standards, and allows one to fit a model to a specific network, which is of great interest to track evolving architectures. We call this process the *self-modeling*. This task can indeed be automatised provided the management framework provides tools to explore the network and reveal its architecture. Such a 'service,' or this 'reflectivity property' of the network, is one of the essential functions one should expect from a self-management framework, together with the ability to query the managed elements in order to check some of their state variables or to perform tests.

The network model obtained by this approach may be huge, in particular if one has to capture all private equipment of all users. But it is not necessary to build it beforehand as a necessary input to an adequate monitoring/diagnosis algorithm. Such an approach would not scale up. We therefore adopt a strategy where only part of the model is built on the fly, in order to solve a specific malfunction. The next chapter formalizes this approach and illustrates it on a simple example.

Chapter 7

Self-Modeling as Support for Fault Localization

Contents

7.1 Methodology **126**

7.2 Generic Bayesian Networks **131**

7.3 Fault localization in a GBN instance **132**

The previous chapter illustrated a typical feature of network design: low-level resources are assembled to build a higher level resource, where the term 'resource' ranges from physical component to high-level capabilities. These complex dependency relations naturally orient us to a formalization in terms of Bayesian networks (BN), which are particularly suited to represent both constraints and statistical dependencies. At the same time they encode conditional independence relations, which constitute the basis of efficient inference algorithms, a topic covered by a vast body of literature. In this context, inference means to infer the value of some state variables given the observed value of other variables.

The framework needs adaptations however, in several directions. Firstly, the managed network evolves in time (users attach, register, de-register, new equipment or functionalities are added, etc.) and may not be known entirely and in full details. So the BN used for inference should be built on request, capturing the state of the network, to answer a given diagnosis query.

Secondly, not all network resources are involved in the malfunction of say some high level capability. So only part of the network should be taken into account.

But, thirdly, this BN model construction should be coupled with the inference engine. Users share some of the network resources, therefore they carry information about their state, which can be useful to the reasoning. So one must design a dynamic construction of the BN modeling the network, or equivalently a dynamic exploration of the network, to collect more information and solve a given diagnosis query.

Finally, let us clarify what this diagnosis query means. Assuming some network resource is observed as down (e.g. IP configuration down for a specific UE), one has to discover the root cause of this failure, which necessarily lies in the lower level resources that are assembled to build the damaged one. This can be understood as a state

inference problem given the values observed on other state variables. By extension, one could imagine querying the state of *all* variables of a given *type*, given some observations. This corresponds to a questions like, for example, given access segment failure, what is the probability that a *non specified UE* be not able to make calls. This type of generic inference query is new in the BN formalism, and is clearly an approach to perform fault impact analysis.

7.1 Methodology

Our approach to troubleshoot an observed malfunctioning resource is based on the following methodology:

1. Find and/or assemble (at some level of abstraction) the generic model (or generic BN) that describes the resources used by the malfunctioning resource. The malfunctioning resource is the entry point of the fault localization query and is generally a capability. But, it can also be any other IMS resource.

2. Locate the instance of this generic BN in the IMS network instance, thus deriving a BN instance, that we call a *pattern*.

3. Within the current BN (instance), feed the inference engine with available observations to locate the faulty resource.

4. If the collected observations are not sufficient, expand the current BN by exploring other patterns (BN instances) that share resources with the current BN. In this expanded BN, or expanded neighborhood of the malfunction in the network, collect the new available observations to improve fault location.

5. Repeat the expansion until confidence in the explanation becomes sufficient.

To illustrate this methodology on a simple practical case, suppose that we want to explain why the IP configuration failed for the user Laurie in the network instance of Figure 7.1.

IP configuration comprises the IP address allocation to the UE, and the discovery of addressing information to the Proxy-Call Session Control Function (P-CSCF). As shown by Figure 7.2, the IP configuration capability is broken down into two successive stages, initiated by the UE.

For clarity, let us select a single granularity of description within each layer. In the functional layer, this granularity distinguishes a single resource denoted 'NACF*' that groups the Access Management Function (AMF), the Network Access Configuration Function (NACF), the User Access Authorization Function (UAAF), the communication interface between AMF and NACF (int AMF - NACF) and the one between NACF and UAAF (int NACF - UAAF). In the same manner, the chosen granularity distinguishes a single resource denoted 'CLF*' that groups the Connectivity session Location and repository Function (CLF), the Access-Resource and Admission Control

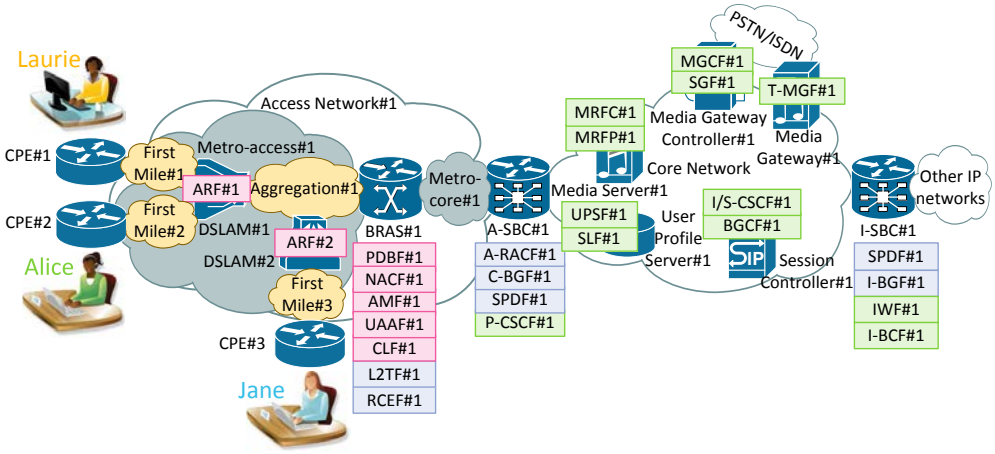


Figure 7.1: An IMS network instance: physical layer, with mention of the supported functions of the functional layer.

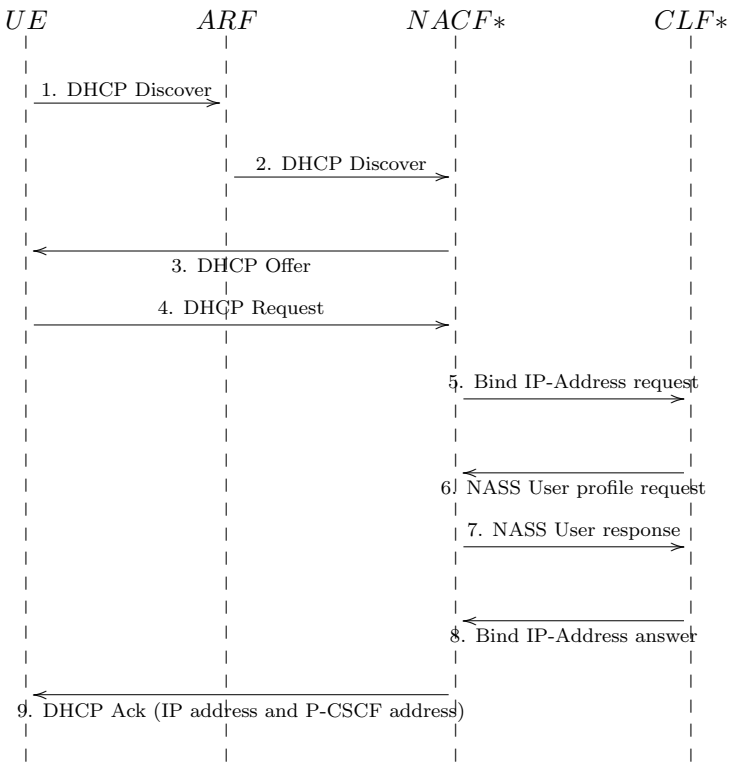


Figure 7.2: Sequence diagram of IP configuration using DHCP (procedural layer). It consists in a dialog between the User Equipment (left) and functions of the NASS (right).

Function (A-RACF) and the communication interface (int CLF - A-RACF) between these two functions.

The UE behaves as a DHCP client and during the first stage, it broadcasts a DHCP Discover message (1) to locate available DHCP servers. The message is received by the NACF* that behaves as a DHCP server, via the ARF that acts as a DHCP relay. Next, the UE receives a DHCP Offer message (3) with an offer of configuration parameters. During the second stage, the UE requests offered parameters with a DHCP Request message (4). This message is relayed towards the NACF* that commits binding and responds with a DHCP Ack message.

Following the methodology described above, we first retrieve the generic model that describes the resources used by the IP configuration capability (Figure 7.3). This dependency graph between resources can be regarded as a generic BN, with statistical dependencies if statistics are available, or logical dependencies otherwise. It encodes, for example, that the outcome of the 'IP configuration' procedure demands that 'Stage 1' be properly performed, which depends on the state of the interface between the functions UE and ARF denoted here as 'int UE-ARF', which in turn uses the communication channel between the UE and the ARF, that is the first mile segment. In this generic BN, the state variable 'IP configuration' is observable, since one can easily test whether the UE obtained a correct IP address.

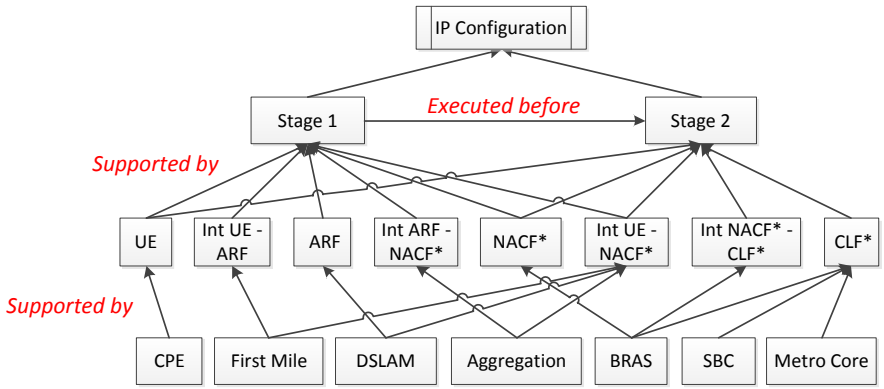


Figure 7.3: Generic BN for the IP configuration capability of a User Equipment.

As a second step, we locate among multiple instances of this generic BN, the instance that refers to the user Laurie. This BN instance is shown in Figure 7.4. The other BN instances refer to the other users: Jane and Alice. In Lauries's BN instance, some resources are private to Laurie. Such resources are displayed in orange to distinguish them from the resources that are shared with other users. The state of the observable variable 'IP configuration' in the BN instance attached to Laurie is set to 'down,' *i.e.* not functioning correctly.

Any node in Laurie's BN instance can be an explanation for the observed malfunction. The third step consists in collecting the available observations on all these resources, and running a BN inference, to try and locate the origin of the malfunction.

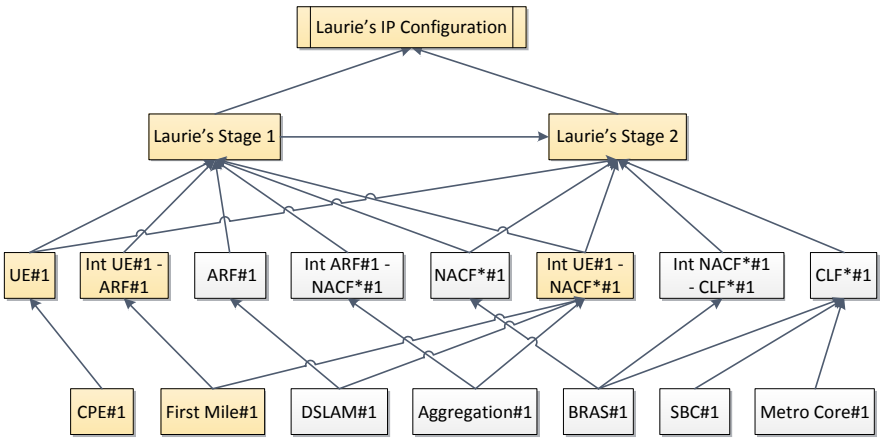


Figure 7.4: Laurie's BN instance

Not all resources may provide observable state information. And some observation may not be fully discriminant. For example, one may observe that the Aggregation segment is in state 'loaded', but may not be certain that this is the origin of the failure. There is simply some probability that this caused a crash of the IP configuration, for example because of the time-outs in the DHCP protocol.

As a fourth step, one may decide to cross-check the possible explanations discovered so far by querying other users that share some resources with Laurie. For example, we choose to check the state of Jane's IP configuration. Thus, we extend the scope of the BN under study (Laurie's BN instance) by adding Jane's BN instance. Figure 7.5 shows the extended BN, which now incorporate more observable variables. Suppose that the state of Jane's IP configuration is 'up', functioning correctly. This observation implies that all the resources Laurie shares with Jane are working properly (which would have been revealed by a BN inference on this extended BN). As a consequence, the set of possible faulty resources in Laurie's BN instance is reduced to the subset of resources that are not shared with Jane.

This process can be iterated (point 5 in Section 7.1): one may query another user, provided he/she shares resources with either Laurie or Jane. But the choice of the user to query should depend on how informative the new observations will be to locate the faulty resource that caused Laurie's problem. If we choose to query Alice, who has more resources in common with Laurie than Jane has, the current BN must again be extended to incorporate Alice's BN instance of the IP configuration generic model. Figure 7.6 shows the extended BN (due to space limitation Jane's private resources are not displayed). Suppose that Alice's IP configuration is 'up'. This observation implies that all the resources Laurie shares with Alice are functioning. Once more, the set of possible faulty resources in Laurie's BN instance is reduced to the subset of resources that are not shared with Alice or Jane. By contrast, suppose that Alice's IP configuration is 'down'. This observation increases our confidence in the state 'down'

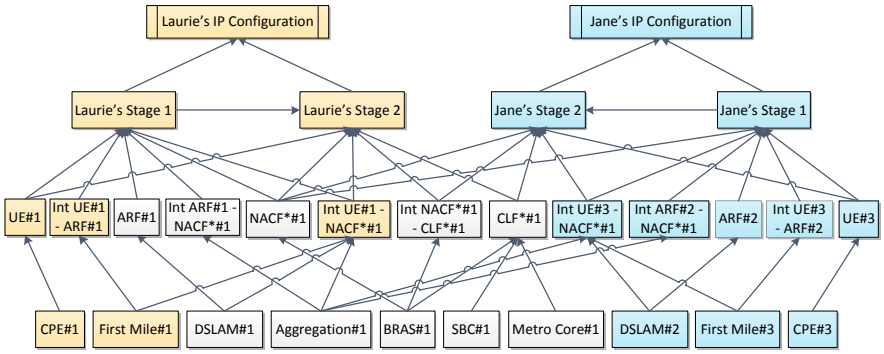


Figure 7.5: Jane's BN instance is added; her private resources are represented in blue

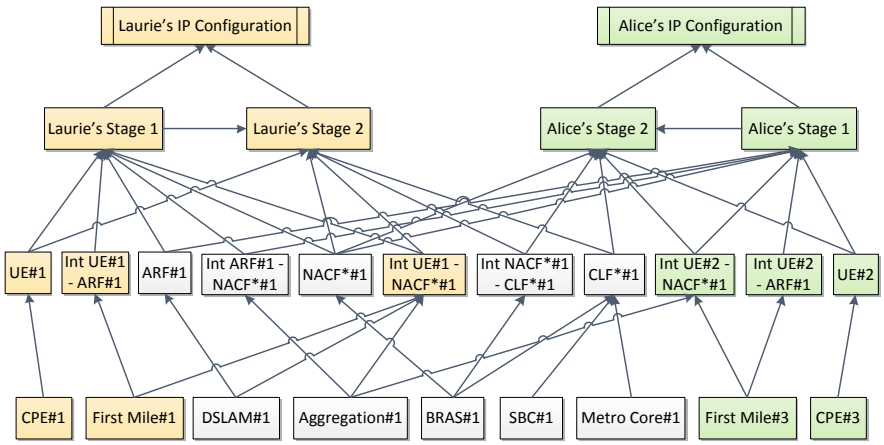


Figure 7.6: Alice's BN instance is added; her private resources are represented in green. Jane's private resources are not shown.

for the resources in Laurie's BN instance that are shared with Alice, and not shared with Jane. The cause of Laurie's IP configuration failure is likely to be found among these shared resources.

7.2 Generic Bayesian Networks

The previous section has demonstrated how the resources involved in a network and its services depend on each other, thus forming a huge dependency graph. This graph can be modeled as a Bayesian network where many parts are isomorphic, *i.e.* this graph is obtained by connecting tiles that are copies of a limited family of generic patterns. This section formalizes this construction of a possibly large Bayesian network, and explains how to perform inference over it, by exploring only the portion of the BN that is the most informative to a given diagnosis query.

Definition 8 *A Bayesian network (BN) $X = (V, E, \mathbb{P}_X)$ is formed of a finite directed acyclic graph (DAG) $G = (V, E)$, with V as vertex set and $E \subseteq V \times V$ as edge set, and a collection of random variables $(X_v)_{v \in V}$ indexed by V and with probability distribution $\mathbb{P}_X$. Denoting ${}^\bullet v = \{u \in V : (u, v) \in E\}$ the parents of node $v \in V$ in the DAG (V, E), the distribution of X factorizes as $\mathbb{P}_X = \otimes_{v \in V} \mathbb{P}_{X_v | X_{{}^\bullet v}}$, where $X_U = (X_u, u \in U)$ denotes a vector of random variables, $U \subseteq V$.*

As usual, a morphism $\phi : G_1 \to G_2$ between DAGs $G_i = (V_i, E_i)$ is a partial function from V_1 to V_2 preserving the edges: $\forall u, v \in Dom(\phi) = V_1' \subseteq V_1$, $(u, v) \in E_1 \Leftrightarrow (\phi(u), \phi(v)) \in E_2$. ϕ is said to be an insertion of G_2 into G_1 iff ϕ restricted to its domain is bijective, *i.e.* $\phi_{|V_1'}$ is an isomorphism between $G_{1|V_1'} = (V_1', E_1 \cap V_1' \times V_1')$ and G_2.

Definition 9 *A Generic Bayesian network (GBN) is a finite collection of ordinary BNs $(W^k)_{1 \leq k \leq K}$, where each BN $W^k = (V_k, E_k, \mathbb{P}_{W^k})$ is also called a pattern. An instance $X = (V, E, \mathbb{P}_X, (\phi_{k,i})_{k \leq K, i \in I_k})$ of this GBN is a standard BN $(V, E, \mathbb{P}_X)$ where*

1. *each $\phi_{k,i}$ is an insertion of pattern (V_k, E_k) into (V, E), and the DAG (V, E) is covered by such pattern instances: $V = \cup_{k,i} Dom(\phi_{k,i})$, $\forall (u, v) \in E$, $\exists \phi_{k,i} : u, v \in Dom(\phi_{k,i})$.*

2. *probability $\mathbb{P}_X$ is inherited from patterns W_k through the insertion morphisms $(\phi_{k,i})_{k \leq K, i \in I_k}$: $\forall v \in V$, one has*

   a) *if ${}^\bullet v = \emptyset$, then $\forall k, i : \phi_{k,i}(v) = u \Rightarrow \mathbb{P}_{X_v} \equiv \mathbb{P}_{W^k_u}$*
   b) *if ${}^\bullet v \neq \emptyset$, then $\forall k, i : \phi_{k,i}(v) = u, {}^\bullet v \cap Dom(\phi_{k,i}) \neq \emptyset \Rightarrow {}^\bullet v \subseteq Dom(\phi_{k,i})$, $\mathbb{P}_{X_v | X_{{}^\bullet v}} \equiv \mathbb{P}_{W^k_u | W^k_{{}^\bullet u}}$*

In other words, to build X one takes many copies of the different patterns W^k, and aggregates them by sharing some of the random variables. To ensure consistency of this construction, each variable X_v (with parents) appearing in several pattern instances

must be defined by the same conditional probability. So either X_v comes alone in an instance, or it comes with all its parents, and in this case the same $\mathbb{P}_{X_v|X_{\bullet v}}$ is obtained from the different patterns. A similar condition holds when X_v has no parent. Figure 7.7 gives an example of a GBN with a single pattern of five variables, and an instance of this GBN with six copies of the pattern, overlapping in different ways. Observe that a given variable may play different 'roles' according to the pattern instance where it is considered.

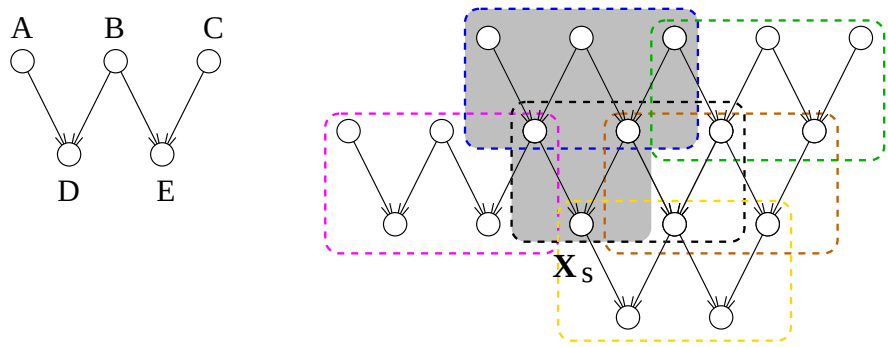


Figure 7.7: A GBN (left) with a single pattern, and an instance of this GBN (right) with six overlapping instances of this pattern.

7.3 Fault localization in a GBN instance

Fault localization in a GBN instance can be translated into a standard BN inference as follows. Assume some variable, for example X_s in Figure 7.7, is declared/observed as faulty. This resource X_s depends on other resources, namely $X_{\bullet\bullet s}$ where ${}^{\bullet\bullet}s$ denotes all the ancestors of node s. Either the failure of X_s is spontaneous, or it results from a fault propagation pattern in $X_{\bullet\bullet s}$. The objective is thus to estimate the value of X_Z, with $Z = {}^{\bullet\bullet}s \cup \{s\}$ (in gray in Figure 7.7). Alternatively, one can build a root-cause localizer L, as a function of X_Z that constrains the presence of a single root-cause within X_Z. The objective is then to estimate the value of L.

There exist observable variables within (or directly attached to) X_Z, in particular the observation indicating the faulty value of X_s. Let us denote by $Y_0 = y_0$ this observed vector. The conditional law $\mathbb{P}_{X_Z|Y_0=y_0}$ gives a first information of what caused the failure of X_s: by maximum likelihood estimation, one gets an explanation (*i.e.* a fault propagation scheme) as the value $\hat{X}_Z = \arg\max_x \mathbb{P}_{X_Z|Y_0=y_0}(x)$. How reliable this explanation is can be determined by computing the conditional entropy of X_Z given $Y_0 = y_0$:

$$H(X_Z|Y_0 = y_0) = \sum_x -\mathbb{P}_{X_Z|Y_0=y_0}(x) \cdot \log_2 \mathbb{P}_{X_Z|Y_0=y_0}(x)$$

A large conditional entropy means that the observations $Y_0 = y_0$ are insufficient to discriminate among several explanations. One may therefore seek to collect more

observations in order to increase more information about X_Z, that is reduce the conditional entropy: the closer to zero, the most reliable the estimation of X_Z. The idea is the to explore a larger area of the BN instance in order to collect more observations. From the explored set of nodes $U_0 = Z$ one thus goes to $U_1 \supseteq U_0$ such that U_1 is closed for the ancestor relation: ${}^{\bullet}U_1 \subseteq U_1$. This defines a new set of observations $Y_1 = y_1$, from which $\hat{X}_Z$ can be estimated provided $H(X_Z|Y_0 = y_0, Y_1 = y_1)$ is small enough. Otherwise one proceeds to another extension.

What is the rationale governing the BN instance exploration ? Referring to Figure 7.7, one may extend the explored part of the BN instance to capture observations in either the green pattern instance, or those in the magenta one. Let us denote Y_1 and Y_2 these two possible sets of observations. The most informative one is obtained by comparing $H(X_Z|Y_0 = y_0, Y_1)$ to $H(X_Z|Y_0 = y_0, Y_2)$: the smaller one wins, which selects the set of observations that is the most informative *on the average*. This is computed beforehand, without querying/testing the actual value of the selected Y_i. Assume Y_1 was the most promising set of measurements, this defines the new area U_1 taken into account in the BN instance, where one may collect the observed value $Y_1 = y_1$. After standard inference, this yields the new posterior distribution $\mathbb{P}_{X_Z|Y_0=y_0,Y_1=y_1}$ and thus the conditional entropy $H(X_Z|Y_0 = y_0, Y_1 = y_1)$. Notice that $H(X_Z|Y_0 = y_0, Y_1 = y_1)$ may actually be either smaller or larger than the averaged value $H(X_Z|Y_0 = y_0, Y_1)$ used to determine the most promising set of observations.

The introduction of new measurements proceeds until either the conditional entropy is small enough, or no more observations are available. Notice that by construction of BN, the exploration is necessarily performed by exploring pattern instances that share variables with the explored section of the BN. Measurements lying in disconnected areas are independent of X_Z, and thus of no help to estimate it. Observe also that such a recursive exploration approach opens the way to a tradeoff when the collection of new observations incurs some cost, for example if tests must be performed. One can therefore balance the amount of information that can be gained with the cost of collecting it.

Chapter 8

Experimental results

Contents

8.1 Entropy, relative entropy and mutual information 135

8.1.1 Entropy and conditional entropy 135

8.1.2 Relative entropy and mutual information 136

8.1.3 Gain function for fault localization 138

8.2 Implementation and evaluation 139

8.2.1 Entropy reduction as more measurements are collected 139

8.2.2 Fault localization in an IMS network 141

The aim of this chapter is to demonstrate the relevance of the exploration strategy explained in the previous chapter. We review some standard results and definitions related to entropy, relative entropy, mutual information and gain function for fault localization. These concepts and their properties are useful to analyse the experimental results that will be presented.

8.1 Entropy, relative entropy and mutual information

For any probability distribution, we recall the definition of a quantity called the *entropy*, which has many properties that agree with the intuitive notion of what a measure of information should be. This notion is extended to define *mutual information*, which is a measure of the amount of information one random variable contains about another. Entropy then becomes the self-information of a random variable. Mutual information is a special case of a more general quantity called *relative entropy*, which is a measure of the distance between two probability distributions. All these quantities are closely related and share a number of simple properties that are presented below.

8.1.1 Entropy and conditional entropy

We will first introduce the concept of entropy, which is a measure of uncertainty of a random variable. Let X be a discrete random variable with alphabet $\mathscr{X}$ and probability mass function $p(x) = Pr(X = x)$, $X \in \mathscr{X}$. We denote the probability mass function by $p(x)$.

Definition 10 *The entropy $H(X)$ of a discrete random variable X is defined by*

$$H(X) = -\sum_{X \in \mathscr{X}} p(x) log\, p(x). \tag{8.1}$$

Note that $H(X) \geq 0$.

We also recall the conditional entropy of a random variable given another as the expected value of the entropies of the conditional distributions, averaged over the conditioning random variable.

Definition 11 *If $(X, Y) \sim p(x, y)$, then the conditional entropy $H(Y|X)$ is defined as*

$$\begin{aligned} H(Y|X) &= \sum_{X \in \mathscr{X}} p(x) H(Y|X = x) && (8.2) \\ &= -\sum_{X \in \mathscr{X}} p(x) \sum_{Y \in \mathscr{Y}} p(y|x)\, log\, p(y|x) && (8.3) \\ &= -\sum_{X \in \mathscr{X}} \sum_{Y \in \mathscr{Y}} p(x, y)\, log\, p(y|x). && (8.4) \end{aligned}$$

The naturalness of the definition of joint entropy and conditional entropy is exhibited by the fact that the entropy of a pair of random variables is the entropy of one variable plus the conditional entropy of the other.

Theorem 1 (Chain rule for entropy)

$$H(X, Y) = H(X) + H(Y|X). \tag{8.5}$$

Let $X_1, X_2, ..., X_n$ be drawn according to $p(x_1, x_2, ..., x_n)$. Then

$$H(X_1, X_2, ..., X_n) = \sum_{i=1}^{n} H(X_i|X_{i-1}, ..., X_1). \tag{8.6}$$

8.1.2 Relative entropy and mutual information

Here, we review two related concepts: relative entropy and mutual information.

Definition 12 *The relative entropy or Kullback Leibler distance between two probability mass functions p(x) and q(x) is defined as*

$$D(p\|q) = \sum_{X \in \mathscr{X}} p(x) log\, \frac{p(x)}{q(x)}. \tag{8.7}$$

Definition 13 *Consider two random variables X and Y with a joint probability mass function $p(x, y)$ and marginal probability mass function $p(x)$ and $p(y)$. The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution $p(x)p(y)$:*

$$I(X;Y) = \sum_{X \in \mathscr{X}} \sum_{Y \in \mathscr{Y}} p(x,y) \, log \, \frac{p(x,y)}{p(x)p(y)} \tag{8.8}$$

$$= D(p(x,y) \| p(x)p(y)) \tag{8.9}$$

$$= H(X) - H(X|Y). \tag{8.10}$$

Thus, the mutual information $I(X;Y)$ is the reduction in the uncertainty of X due to the knowledge of Y. Using Jensen's inequality and its consequences (Cover and Thomas, 1991) some of the properties of entropy and relative entropy can be proved.

Theorem 2 (Information inequality) *Let $p(x)$, $q(x)$, $x \in \mathscr{X}$, be two probability mass functions. Then,*

$$D(p \| q) \geq 0 \tag{8.11}$$

Corollary 4 (Non-negativity of mutual information) *For any two random variables, X, Y:*

$$I(X;Y) \geq 0, \tag{8.12}$$

Theorem 3 (Conditioning reduces entropy)

$$H(X|Y) \leq H(X) \tag{8.13}$$

Intuitively, the theorem says that knowing another random variable Y can only reduce the uncertainty in X. Note that this is true **only on the average**. Specifically, $H(X|Y = y)$ may be greater or lower than $H(X)$, but on the average $H(X|Y) = \sum_y p(y)H(X|Y = y) \leq H(X)$. For example, in a court case, specific new evidence might increase uncertainty, but on the average evidence decreases uncertainty.

Example. Let (X, Y) have the following joint distribution. Then, $H(X) = H(\frac{1}{8}, \frac{7}{8}) =$

	X=1	X=2
Y=1	0	$\frac{3}{4}$
Y=2	$\frac{1}{8}$	$\frac{1}{8}$

0.544 bits, $H(X|Y = 1) = 0$ bits and $H(X|Y = 2) = 1$ bit. We calculate $H(X|Y) = \frac{3}{4}H(X|Y = 1) + \frac{1}{4}H(X|Y = 2) = 0.25$ bits. Thus, uncertainty in X is increased if $Y = 2$ is observed and decreased if $Y = 1$ is observed, but uncertainty decreases on the average. Notice again that this information reduction $H(X|Y) \leq H(X)$ is on the average. For a given sample, one may have $H(X|Y = y) \geq H(X|Y)$ and even $\geq H(X)$.

Theorem 4 (Chain rule for information)

$$I(X;Y_1,Y_2,...,X_n) = I(X;Y_1) + I(X;Y_2|Y_1) + I(X;Y_3|Y_1Y_2) + ... \tag{8.14}$$

In particular,

$$H(X|Z=z,Y=y) = -\sum_x p(x|y,z) log p(x|y,z). \tag{8.15}$$

$$H(X|Z,Y=y) = \sum_z p(X|Z=z,Y=y)\dot{p}(z|y). \tag{8.16}$$

so that, to select the next measurement Z, we need $p(z|y)$ and $p(x|y,z)$ or equivalently $p(x,z|y)$ (i.e. $P(X,Z|Y=y)$, the full law for X,Z and fixed y).

8.1.3 Gain function for fault localization

Suppose the state of a system is denoted by a vector $X = (X_1, X_2, ..., X_n)$ of random variables, where X_i represents the state of a node. A probabilistic model is a joint probability distribution (a prior distribution) over all the random variables X_i. Hence it assigns, a probability $p(x)$ to each realization x. Given observations Y of the system state, a fault diagnosis algorithm identifies the Most Probable Explanation (MPE) of the underlying system responsible for observed outage, i.e.:

$$x^* = \arg\max_x p(x|y)$$

Observations reduce uncertainty of the system. Before we observe Y, the uncertainty about system state can be quantified by the Shannon entropy:

$$H(X) = -\sum_x p(x)\, log\, p(x).$$

If the system state is doubtless, $H(X) = 0$. As uncertainty goes up, $H(X)$ increases. The purpose of making observations is just to reduce this uncertainty. Given probe responses $Y = (Y_1, Y_2, ..., Y_n)$, the probability of the system state changes into $P(X|Y)$, consequently the average uncertainty of the system state is:

$$H(X|Y) = -\sum_{x,y} p(x,y)\, log\, p(x|y).$$

(Lindley, 1956) proposed a method to measure the average information provided by an experiment. Suppose the probe Y_i is defined on system state X.

Definition 14 *The average amount of information gain from Y_i, with prior probability distribution $P(X)$, is:*

$$G(Y_i) = I(X;Y_i) = H(X) - H(X|Y_i) = \sum_{x,y_i} p(x,y_i)\, log\, \frac{p(x,y_i)}{p(x)P(y_i)}. \tag{8.17}$$

In some situations, as the one we examine below, one may not be able to use directly the full observation set $Y = (Y_1, Y_2, ..., Y_n)$ and may have to select a subset of k observations to collect. In order to select an optimal set of k probes among variables $Y_1, ..., Y_n$, for fault localization, the target is to find a subset $E \subseteq \{1, ..., n\}$, $|E| = k$, which maximizes the reduction of Shannon entropy i.e.:

$$E^* = \arg\max_E G(Y_E), \text{ for } Y_E = \{Y_i,\, i \in E\}. \tag{8.18}$$

The problem stated in (8.18) is NP-hard (Brodie et al., 2003), hence efficient exact solutions are likely not possible. An alternative, widely used in the literature, is the heuristic greedy approach, which iteratively adds to E the probe that reduces the maximum entropy of the system, among those that were not selected yet. When problems are detected, a greedy algorithm repeatedly selects, given E, the new observation Y_i to collect by:

$$i^* = \arg\max_{i \notin E} I(X; Y_i | Y_E). \tag{8.19}$$

Actually $i \notin E$ can be simplified into $i \in E$ because $I(X; Y_i|Y_E) \geq 0$ and $I(X; Y_i|Y_E) = 0$ for $i \in E$. Besides, note that this solution uses $P(X, Y_i|Y_E)$. Not only the mutual information of X and Y_i is measured, but the impact of previously selected probes on Y_i is also calculated, which provides a "global view", better than the techniques which only consider system state X and the candidate probe Y_i.

8.2 Implementation and evaluation

Kevin Murphy's Bayes Net Toolbox (Murphy et al., 2001) is an open-source Matlab package for directed acyclic graph (DAG) models, also known as Bayesian or belief networks. It has proved to be very popular and it is widely used throughout the world for teaching and research. Bayes Net Toolbox (BNT) supports many kind of nodes (probability distributions) and offers a variety of exact and approximate inference algorithms. We use the junction tree inference engine in BNT to implement our exploration strategy.

8.2.1 Entropy reduction as more measurements are collected

We have first performed some tests on a large tree-shaped GBN instance. The (unique) pattern corresponds to Figure 8.1 where variables X_D and X_E are observable. The instance is depicted in Figure 8.2, where each vertex represents a pattern instance (thus 5 variables). Two connected pattern instances overlap by 1, 2 or 3 of the $\{A, B, C\}$ nodes, which is reflected by the thickness of the edge relating these patterns. In pattern (instance) X_1, variables $X_{1,A}, X_{1,B}, X_{1,C}$ are (independent) uniform binary variables. In all other patterns X_i, $i > 1$, the newly created variables $X_{i,A}, X_{i,B}$ or $X_{i,C}$ have distribution $(p, 1-p)$ with $p = 0.9$, in order to guarantee long-range correlation between patterns in the net of Figure 8.2. In pattern X_i, the observations Y_i correspond to variables $X_{i,D}, X_{i,E}$. D is more sensitive to a failure of B than to a failure of A, and

similarly E reacts more to C than to B. Both sensors have a misdetection rate and a false alarm rate of 5%.

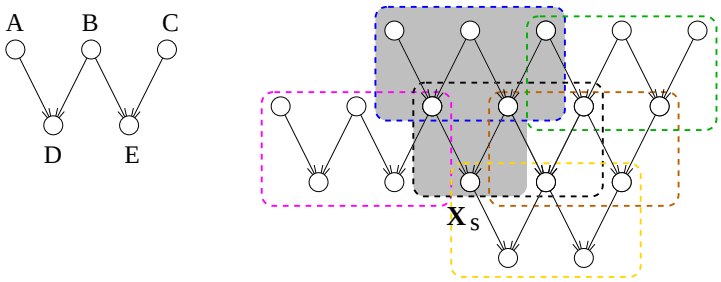


Figure 8.1: A GBN (left) with a single pattern, and an instance of this GBN (right) with six overlapping instances of this pattern.

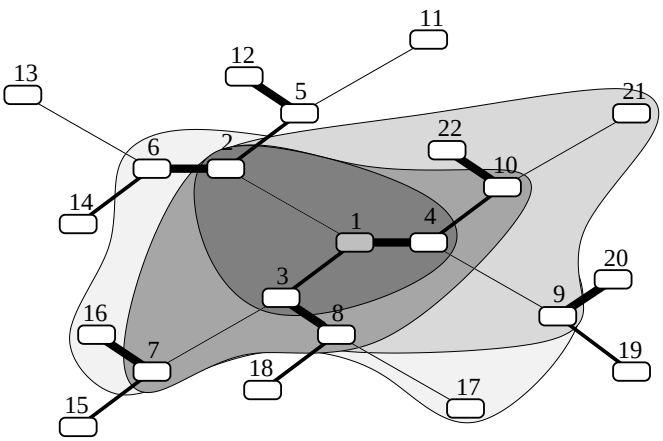


Figure 8.2: A GBN instance with 22 pattern instances. The thickness (and length) of the line relating two patterns reflects the number of shared variables. To estimate the central pattern X_1, observations are introduced by zones, starting by the dark zones first and progressing towards the pale ones.

The experiment consists in drawing a random sample of the process described by this GBN instance, and in computing the conditional distribution $\mathbb{P}_{X_1|Y_U=y_U}$ of the central pattern X_1 given a growing set U of measurements collected on X_1 and on distant patterns. The measurement set starts from $U = \{1\}$ up to $U = \{1, 2, ..., 22\}$ where U is the index set of the patterns where observation are collected. In between, the most informative measurement is incorporated at each step. For each sample, the evolution of $H(X_1|Y_U = y_U)$ was computed. The experiment was conducted one thousand times. On the average, one observes that $H(X_1|Y_U = y_U)$ does decrease (Figure 8.3), and quickly converges. However, for a given sample, this curve may not always decrease: even if an observable node is very informative on the average, its actual observed sample may lead to a revision of the current assumptions, thus increasing temporarily

the uncertainty. An example of such a non-decreasing behaviour is presented later in this chapter.

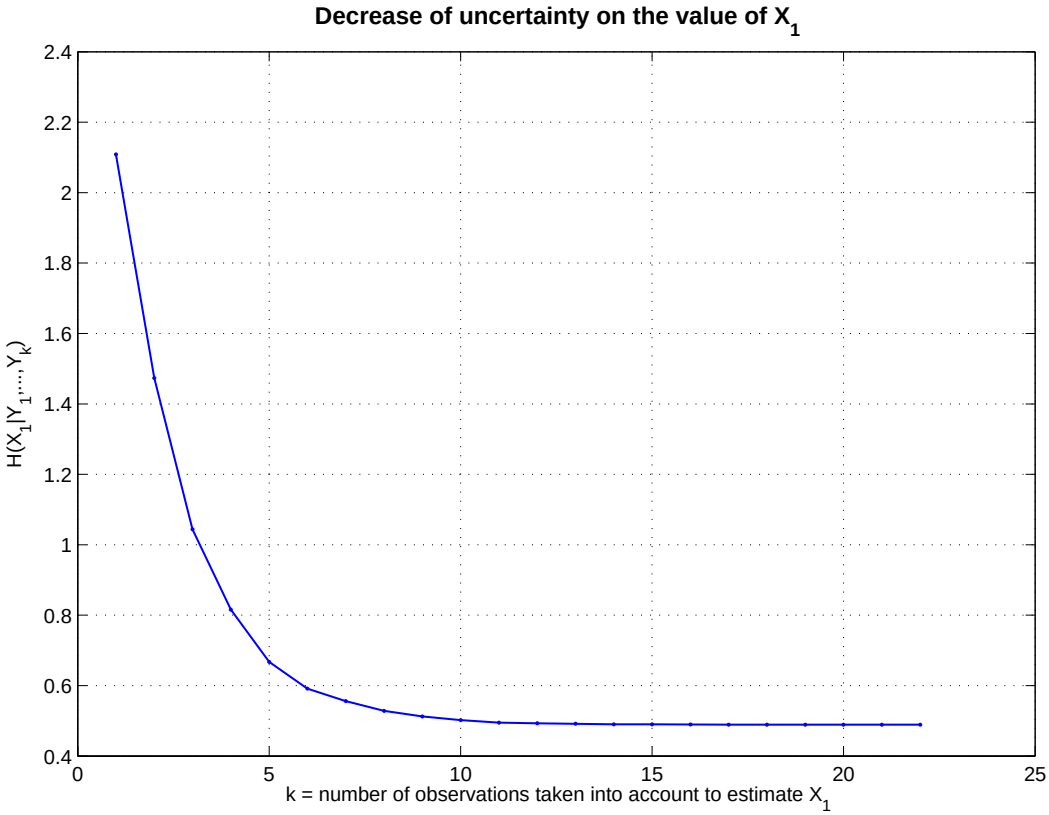


Figure 8.3: Average evolution of the conditional entropy of the central pattern X_1 as the number of observations increases.

More interestingly, we have performed rank statistics on the way measurements are introduced. On the average, the best order appears to be $1, 3, 4, 2, 7, 10, 8, 9, 21, 22, 17, 6, 16, 5, 13, 15, 20, 18, 19, 12, 14, 11$, which is reflected by the growing zones around X_1 in Figure 8.2. Strongly coupled patterns tend to be favored, but not always.

8.2.2 Fault localization in an IMS network

We have then performed another experiment on a more realistic GBN instance. The network instance considered is shown in Figure 8.4.

Description of the failure scenario

Suppose that we want to explain why the IP connectivity is down for the user Mary in the network instance of Figure 8.4.

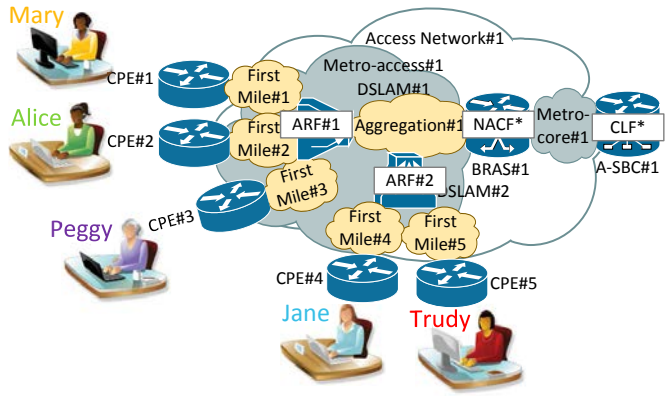


Figure 8.4: Network instance example: the physical and functional layers at a given resolution.

As shown by Figure 8.5, the IP connectivity procedure is broken down into two successive procedures, initiated by the UE. In the functional layer, the chosen granularity

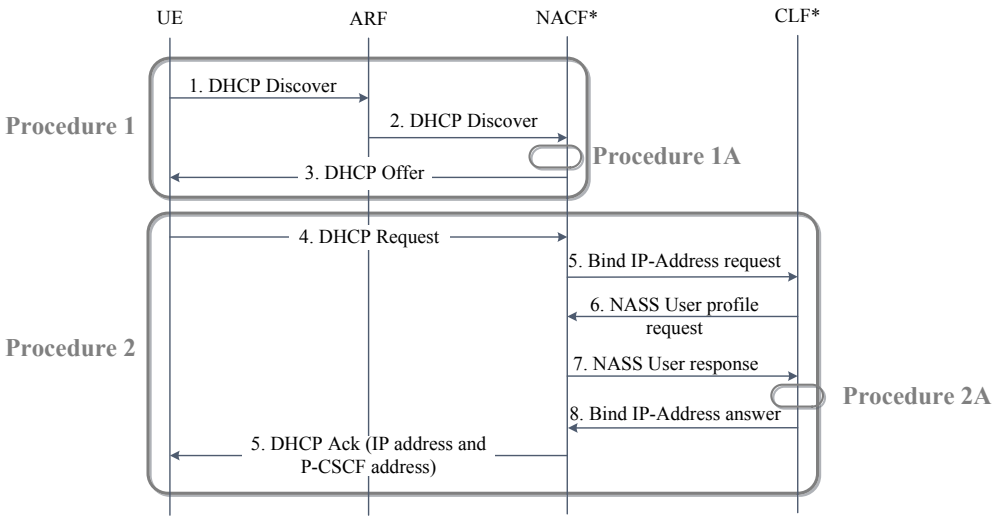


Figure 8.5: Sequence diagram of IP connectivity using DHCP (procedural layer). It consists in a dialog between the User Equipment, functions of the NASS and the RACS.

of description distinguishes a single resource denoted 'NACF*' that groups the Access Management Function 'AMF', the Network Access Configuration Function 'NACF', the User Access Authorization Function 'UAAF', the communication interface between 'AMF' and 'NACF' ('int AMF-NACF') and the one between 'NACF' and 'UAAF' ('int NACF-UAAF'). In the same manner, the chosen granularity distinguishes a single resource denoted 'CLF*' that groups the Connectivity session Location and repository Function 'CLF', the Access-Resource and Admission Control Function 'A-RACF' and

the communication interface ('int CLF-A-RACF') between these two functions.

The (unique) pattern (generic model) that describes the resources used by the IP connectivity procedure is shown in Figure 8.6. The pattern instance relative to the

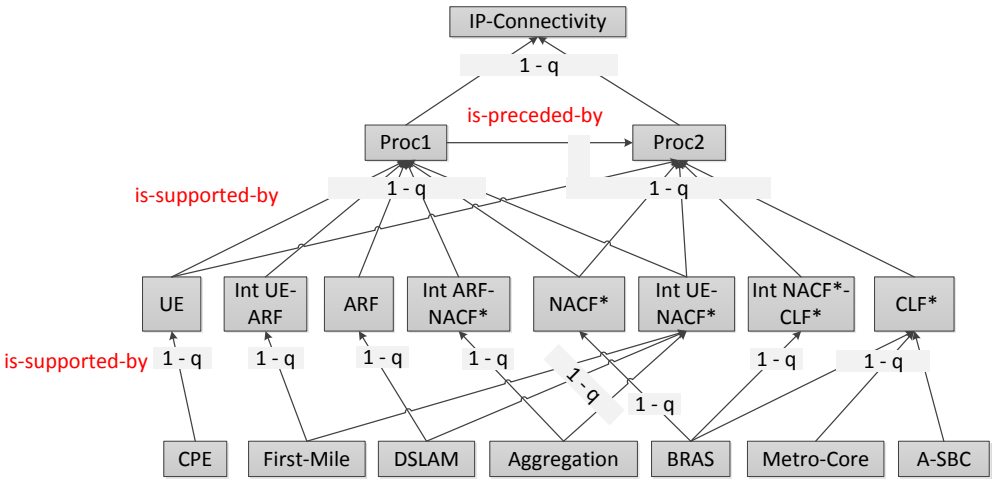


Figure 8.6: Pattern for IP connectivity with conditional probabilities displayed. If all these parents are in state 'up', a child is in state 'up', unless it fails by itself and the probability for that is $q = 0.1$.

user Mary is shown in Figure 8.7. The five pattern instances are depicted in Figure 8.8

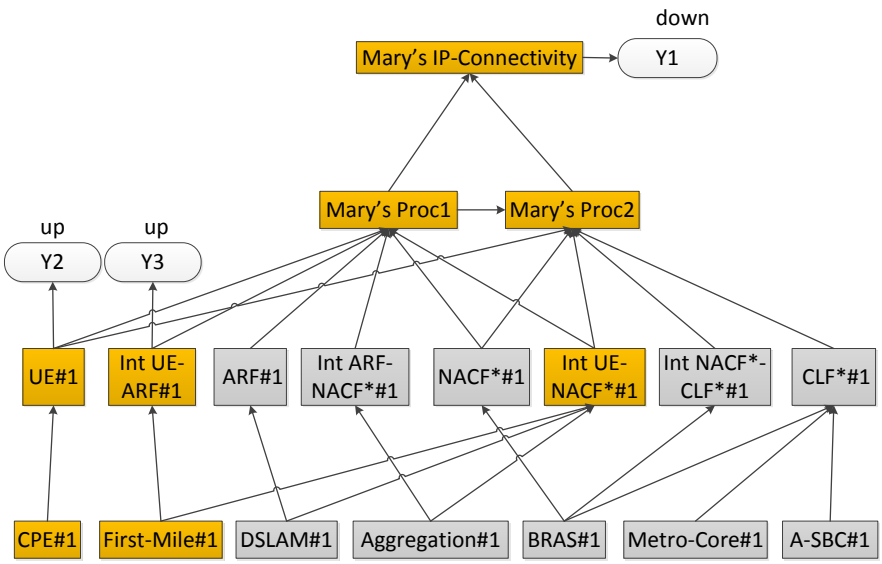


Figure 8.7: Mary's pattern instance. Observable variables are Y_1,Y_2 and Y_3.

where the private resources are distinguishable by colors: Mary's private resources are displayed in orange, Alice's pattern instance in green, Peggy's pattern instance

in purple, Jane's pattern instance in blue, Trudy's pattern instance in red. Shared resources are displayed in gray.

Experimental setup

The parameters for this experiment are described below.

- All variables are binary with states 1 meaning 'down' and 2 meaning 'up'.
- Some of them are observable:
  - Y_1, Y_2, and Y_3 in Mary's pattern instance
  - Y_4 and Y_5 in Alice's pattern instance
  - Y_6 and Y_7 in Peggy's pattern instance
  - Y_8 and Y_9 in Jane's pattern instance
  - Y_{10} and Y_{11} in Trudy's pattern instance

  They are displayed in rounded boxes in Figure 8.8.
- Variables 'CPE', 'First-Mile', 'DSLAM', 'Aggregation', 'BRAS', 'Metro-Core', 'A-SBC' (and their corresponding instances) have no parent, they are (independent) variables with distribution $(0.1, 0.9)$ where 0.1 is the probability of the variable being in state 1 ('down').
- Variables 'UE', 'Int UE-ARF', 'ARF', 'Int ARF-NACF*', 'NACF*', 'Int NACF*-CLF*' (and their corresponding instances) have one parent. See Figure 8.6 for their distribution and Table 8.1 below as an example.

	CPE=1	CPE=2
UE=1	1	0.1
UE=2	0	0.9

Table 8.1: P(UE|CPE)

- Variables 'Int UE-NACF*' and 'CLF*' (and their corresponding instances) have three parents. See Figure 8.6 for their distribution and Table 8.2 below as an example.

	A-SBC=1		A-SBC=2	
	Metro-Core=1	Metro-Core=2	Metro-Core=1	Metro-Core=2
BRAS=1	(1,0)	(1,0)	(1,0)	(1,0)
BRAS=2	(1,0)	(1.0)	(1,0)	(0.1,0.9)

Table 8.2: P(CLF* | BRAS, Metro-Core, A-SBC). The numbers (x,y) in the table represent (CLF*=1,CLF*=2)

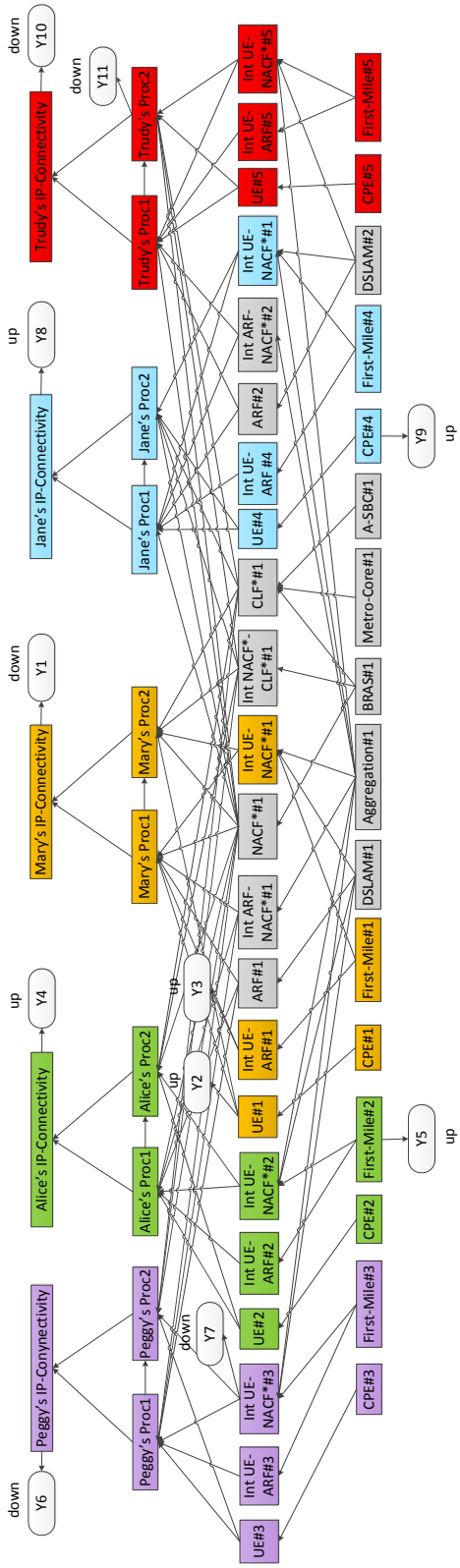


Figure 8.8: The whole model.

- Variables 'Proc1' and 'Proc2' (and their corresponding instances) have six parents. Their distribution is such that if at least one parent is in state down, then they are certainly in state 1 ('down'). When all parents are in state 2 ('up'), these variables can be in the state 1 ('down') with probability 0.1 or in the state 2 ('up') with probability 0.9.

- Variable 'IP-Connectivity' (and its corresponding instance) See Figure 8.6 for their distribution and Table 8.3 below as an example.

	proc1=1	proc1=2
proc2=1	(1,0)	(1,0)
proc2=2	(1,0)	(0.1,0.9)

Table 8.3: P(IP-Connectivity|Proc1,Proc2). The numbers (x,y) in the table represent (IP-Connectivity=1,IP-Connectivity=2)

- All variables representing observations have one parent and distribution $(1, 0.02, 0, 0.98)$. This is the case for Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8, Y_9, Y_{10}, Y_{11}. As an example see Table 8.4 below.

	Mary's IP connectivity=1	Mary's IP connectivity=2
Y_1=1	1	0.02
Y_1=2	0	0.98

Table 8.4: P(Y_1 |Mary's IP connectivity)

- The variable Z (not shown here) has 15 parents: all variables in Mary's pattern instance corresponding to a physical or functional resource. Z can take 15 values, each value pointing toward the failed resource in Mary's pattern instance (see Table 8.5). Note that, we could have chosen sixteen values for Z, the sixteenth value meaning that there is no failure. Furthermore, we could have chosen to add two more states for Z : 17 and 18 to denote the fact that 'proc1' and 'proc2' can fail by themselves i.e. even their underlying functional resources are all in state 'up'. For simplicity, here we use only 15 values.

Results

We observed the distribution of Z and its entropy $H(Z)$ when observations (displayed in rounded boxes in Figure 8.8) are progressively added.

1. When the set of observations $\{Y_1 = 1, Y_2 = 2, Y_3 = 2\}$ in Mary's pattern instance is entered and propagated, we have $H(Z) = 3.3186$ (see Figure 8.9(a) for Z's distribution).

2. Observation Y_7 in Peggy's pattern instance is selected as the most informative one. Peggy's pattern instance is considered and observations $Y_6 = 1$ and $Y_7 = 1$ are

Z's value	Meaning
1	CPE#1 failed
2	First-Mile#1 failed
3	DSLAM#1 failed
4	Aggregation#1 failed
5	BRAS#1 failed
6	Metro-Core#1 failed
7	A-SBC#1 failed
8	UE#1 failed
9	Int UE-ARF#1 failed
10	ARF#1 failed
11	Int ARF#1-NACF*#1 failed
12	NACF*#1 failed
13	Int UE#1-NACF*#1 failed
14	Int NACF*#1-CLF*#1 failed
15	CLF*#1 failed

Table 8.5: Possible values of Z and their respective meaning

entered and propagated. The uncertainty (about the faulty resource) decreased: $H(Z)$ dropped to 3.2575 (see Figure 8.9(b) for Z's distribution).

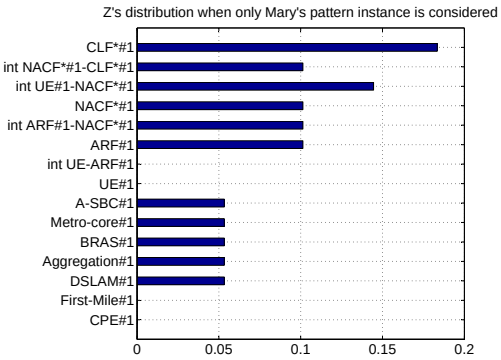
3. Observation Y_{11} in Trudy's pattern instance is selected as the most informative one. Trudy's pattern instance is considered and observations $Y_{10} = 1$ and $Y_{11} = 1$ are entered and propagated. The uncertainty (about the faulty resource) increased: $H(Z)$ went up to 3.2776 (see Figure 8.9(c) for Z's distribution).

4. Observation Y_8 in Jane's pattern instance is selected as the most informative one. Jane's pattern instance is considered and observations Y_8 and Y_9 are entered and propagated. The uncertainty (about the faulty resource) decreased: $H(Z)$ dropped to 1.8529 (see Figure 8.9(d) for Z's distribution).

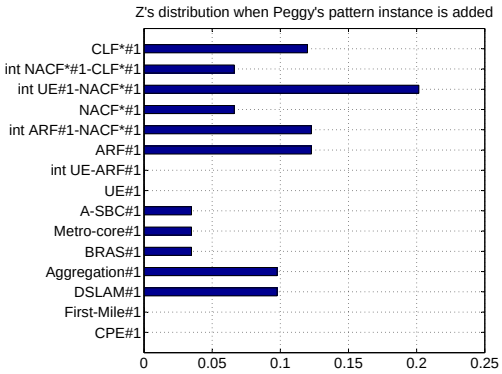
5. Observation Y_4 in Alice's pattern instance is selected as the most informative one. Finally, Alice's pattern instance is considered and observations Y_4 and Y_5 are entered and propagated. $H(Z)$ becomes null meaning that the state of Z i.e. the information about the guilty resource is now doubtless (see Figure 8.9(e) for Z's distribution).

We conduct another experiment consisting in drawing a random sample of the process described by this GBN instance, and in computing the conditional distribution $\mathbb{P}_{Z|Y_i=y_i}$ of the variable Z given the set $Y_1...Y_{11}$ of measurements. The measurement set starts from $V = \{Y_1, Y_2, Y_3\}$ up to $V = \{Y_1, ..., Y_{11}\}$. For each sample, the evolution of $H(Z|Y_V = y_V)$ was computed. The experiment was conducted one thousand times. On the average, one observes that $H(Z|Y_V = y_V)$ does decrease (Figure 8.10).

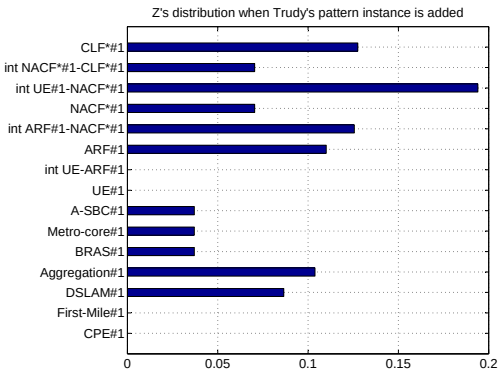
However, for a given sample, this curve may not always decrease, as mentioned above: even if an observable node is very informative on the average, its actual observed sample may lead to a revision of the current assumptions, thus increasing temporarily the uncertainty. For example, this situation happens with the following two samples.



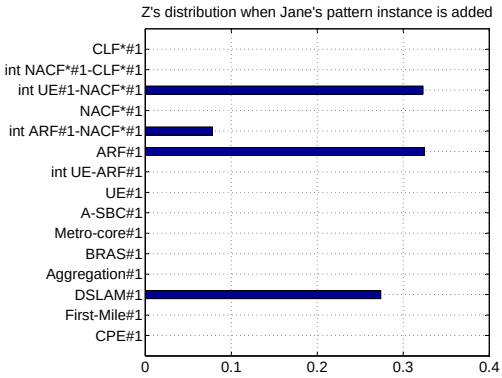
(a) Mary's pattern instance is considered.



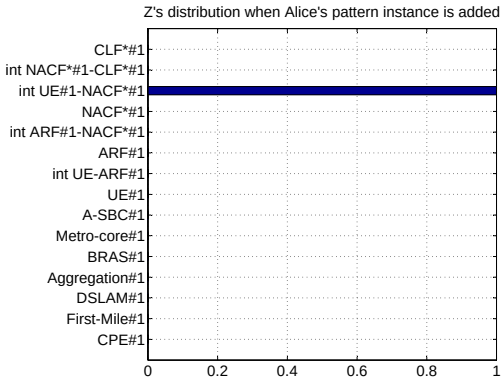
(b) Peggy's pattern instance is added.



(c) Trudy's pattern instance is added.



(d) Jane's pattern instance is added.



(e) Alice's pattern instance is added.

Figure 8.9: Evolution of H(Z) while pattern instances are added

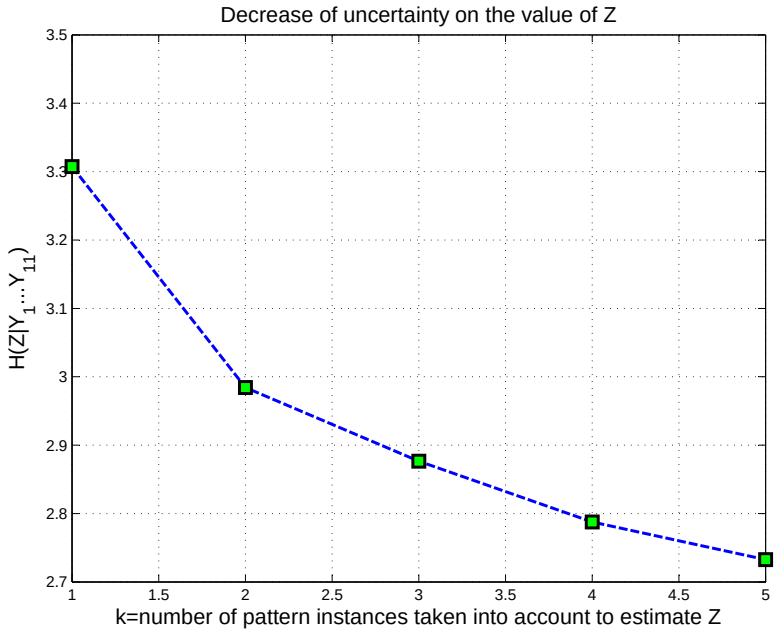
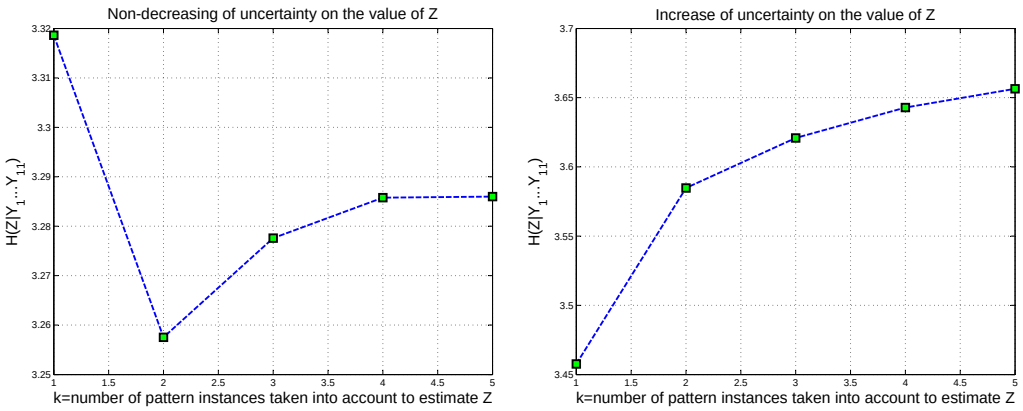


Figure 8.10: Average evolution of the conditional entropy of Z as the number of pattern instances increases.

The first one (sample 1) is characterised by $\{Y_1 = 1, Y_2 = 2, Y_3 = 2, Y_4 = 1, Y_5 = 1, Y_6 = 1, Y_7 = 1, Y_8 = 1, Y_9 = 2, Y_{10} = 1, Y_{11} = 1\}$ (see Figure 8.11(a) for the relative entropy evolution). The second one (sample 2) is characterised by $\{Y_1 = 1, Y_2 = 1, Y_3 = 1, Y_4 = 1, Y_5 = 2, Y_6 = 1, Y_7 = 1, Y_8 = 1, Y_9 = 2, Y_{10} = 1, Y_{11} = 1\}$ (see Figure 8.11(b) for the relative entropy evolution).



(a) Sample 1 (b) Sample 2

Figure 8.11: Examples of non-decreasing behavior of H(Z) while pattern instances are added

Chapter 9

Conclusion

The first part of the thesis address the problem of the Graceful Restart procedure in OSPF networks. We propose to be less conservative as the standardized procedure by taking full advantage of the separation between the control and forwarding functions. We have shown that it was possible at low complexity to preserve the graceful restart procedure of OSPF routers even if the topology changes during this operation. Such a possibility avoids an abrupt return to a normal OSPF restart, which could disrupt the network by large scale (possibly useless) reroutings. The proposed approach yields a softer transition of behavior, where possible routing loops are detected beforehand, and temporarily patched, until the restarting routers return in function. The case of a single restarting router was completely solved: we have characterized the minimal number of reroutings necessary to correct forwarding loops, and have shown where to place them, all this with low complexity and distributable algorithms.

Future work will address the case of several simultaneous reboots, and examine how to deal with successive topological changes, i.e. how to move from a set of temporary reroutings to another one.

In the second part of the thesis, we address the self-diagnosis problem with a model-based approach that aims at managing both the access network, the core network, the functional architecture on top of them and the end-to-end services or the use of these services. Diagnosis here isolating the root cause of symptoms whether they are located in the physical, functional or service layer. The existing management solutions handle separately the service layer, the access and the core network segment, which prevents a smart and automatic analysis of the situation, and blocks any correlation of observed events in these domains. Correlating events of these domains may however require some coordination of the dedicated management resources, that may not be always possible, so single layer versions may also be considered.

Model-based techniques are the key to an ambitious and autonomous management of network malfunctions. They are adaptable to network instances, offer accurate and wide range reasoning techniques, suggest methods to analyse the impact of failures and can certainly go as far as suggesting the best mitigation actions. Nevertheless, they have two weaknesses, namely deriving an accurate model and dealing with huge models.

We have proposed a self-modeling principle that limits the burden of model construction to the design of generic patterns of the model, exploiting knowledge available in the standards. The actual model matching a given network instance is then built

by connecting as many copies/instances as necessary of these generic patterns, in order to reproduce the dependency structure between the resources deployed in this network instance. This guarantees a perfect fit of the model.

However, this model instance may not be known entirely once for all and in full details, for example, if it is supposed to represent the full national network of some operator. First of all because the model evolves in time (users appear, connect, register, disconnect, equipment are added, reconfigured, etc.). But also, one rather needs to know or discover only part of the model, the part that is necessary and sufficient to answer a given diagnosis like "explain why this user cannot make calls". So, the idea is to determine the scope and granularity of the model on which inference will be performed, assuming the model is stationary during the time the query is examined. This means that the network configuration is assumed to change very slowly (e.g. connections that are established remain up for a long time) when compared with the time required to carry out the fault diagnosis. Then, the model instance is progressively built/extended, on the fly, to satisfy the needs of the reasoning/diagnosis process. This exploration scope evolves simultaneously with the inference procedure, which will have to determine whether it is relevant to extend the visible part of the model in order to collect more information about a diagnosis query.

We have proposed a formalism to match this self-modeling principle. It is based on a notion of generic Bayesian network: a BN composed of many copies of a few patterns. Even if the model instance is large, one only needs to explore part of it to explain/diagnose some local observation, since variables located far away carry little information about the observed malfunction.

This formalism is not yet complete however, since it does not yet capture the intrinsic hierarchical construction of networks, that is the fact that a network component generally decomposes into a structure of smaller sub-components, and recursively. Such generic and multi-resolution BN do not exist yet, but seem crucial to network management. Indeed, the large number of dependencies among network components (both physical, functional or procedural) makes it reasonable to perform fault localization in a hierarchical fashion. Fault localization would then be performed starting from a macro-view (high level or abstract view) of the problem to select a potential spot of the problem, an then it should focus on the micro-view (low-level or detailed view) of the chosen spot.

Other new and exciting research directions are opened by this approach, for example the simultaneous or successive processing of multiple diagnosis requests.

Another one is the distribution of the reasoning to capture the fact that network segments are generally managed by different operational units. Expliciting distributed reasoning (which is almost 'built in' the BN formalism) would indicate when these units should communicate and what kind of information. Finally, impact analysis can be performed by checking the influence of a given malfunction of all variables of a given *type* in a given generic pattern of the BN.

The next step of this work will of course be to demonstrate the relevance of this approach on typical failure scenarios in a realistic IMS architecture.

List of publications

- (Hounkonnou and Fabre, 2012) Carole Hounkonnou and Eric Fabre. Empowering Self-diagnosis with Self-modeling. In *8th international conference on Network and Service Management (CNSM)*, CNSM 2012, pages 364–370, 22–26 October 2012 in Las Vegas.

- (Hounkonnou and Fabre, 2013) Carole Hounkonnou and Eric Fabre. Enhanced OSPF graceful restart. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, IM 2013, 27–31 May 2013 in Belgium.

- (Hounkonnou et al., 2012) Carole Hounkonnou, Samir Ghamri-Doudane, and Eric Fabre. Détection et correction de boucles de routage dans un réseau de routeurs utilisant un protocole de routage de type SPF. *Europeean Patent*, 2012.

Bibliography

Armen Aghasaryan, Eric Fabre, Albert Benveniste, Renée Boubour, and Claude Jard. Fault Detection and Diagnosis in Distributed Systems: An Approach by Partially Stochastic Petri Nets. *Discrete Event Dynamic Systems*, 8:203–231, 1998.

C. Angeli. Diagnostic Expert Systems: From Expert's Knowledge to Real-Time Systems. *Advanced Knowledge Based Systems (Model, Applications & Search)*, 1:50–73, 2010.

Leliane Nunes De Barros and Marilza Lemos. Model based diagnosis for network communication faults. In *International Workshop on Artificial Intelligence for Distributed Information Networking, AIDIN '99*, pages 57–62. AAAI Press, 1999.

Emmanuel Bertin, Imen Ben Yahia, and Noël Crespi. Modeling IMS services. *Journal of Mobile Multimedia*, 3(2):150–167, 2007.

David Beyer and Richard Ogier. Tabu learning: a neural network search method for solving nonconvex optimization problems. In *IEEE International Joint Conference on Neural Networks*, pages 953–961. IEEE, 1991.

Anastasios Bouloutas, Seraphin Calo, and Allan Finkel. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications*, 42:523–533, 1994.

Anastasios T. Bouloutas, Seraphin B. Calo, Allan J. Finkel, and Irene Katzela. Distributed Fault Identification in Telecommunication Networks. *Journal of Network and Systems Management*, 3:295–312, 1995.

Mark Brodie, Irina Rish, Sheng Ma, and Natalia Odintsova. Active Probing Strategies for Problem Diagnosis in Distributed Systems. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI 2003, 2003.

Simona Brugnoni, Guido Bruno, Roberto Manione, Enrico Montariolo, Elio Paschetta, and Luisella Sisto. An Expert System for Real Time Fault Diagnosis of the Italian Telecommunications Network. In *Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management with participation of the IEEE Communications Society CNOM and with support from the Institute for Educational Services*, pages 617–628, 1993.

P Calhoun, J Loughney, E Guttman, and G Zorn. RFC 3588: Diameter Base Protocol, 2003.

Chuxin Chen, Teresa L. Hollidge, and D. D. Sharma. Localization of troubles in telephone cable networks. In *Proceedings of the eighth annual conference on Innovative applications of artificial intelligence*, IAAI '96, pages 1461–1470. AAAI Press, 1996.

Jaesung Choi, Myungwhan Choi, and Sang-Hyuk Lee. An alarm correlation and fault identification scheme based on OSI managed object classes. In *IEEE International Conference on Communications*, pages 1547–1551, 1999.

Cisco. NSF-OSPF (RFC 3623 OSPF Graceful Restart), 2012. URL `http://cisco.com/en/US/docs/ios/12_0s/feature/guide/gr_ospf.html`.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 1991.

Robert N Cronk, Paul H Callahan, and Lawrence Bernstein. Rule-based expert systems for network management and operations: an introduction. *Network, IEEE*, 2:7–21, 1998.

F Cuervo, N Greene, A Rayhan, C Huitema, B Rosen, and J Segers. RFC 3015: Megaco Protocol Version 1.0, 2000.

A. Darvishan, H. Yeganeh, K. Bamasian, and P. Eghtedari. A Practical NGN Model By Evaluation of Various NGN Solutions and its Conformance with IMS-TISPAN. In *Proceedings of the 4th International Conference on Ubiquitous Information Technologies Applications, 2009.*, ICUT '09, 2009.

JM David and JP Krivine. Three artificial intelligence issues in fault diagnosis: declarative programming, expert systems, and model-based reasoning. In *Fault detection and reliability: knowledge based and other approaches*, pages 19–27, 1987.

Rina Dechter. *Constraint processing.* Morgan Kaufmann, 2003.

Robert H. Deng, Aurel A. Lazar, and Weiguo Wang. A Probabilistic Approach to Fault Diagnosis in Linear Lightwave Networks. In *Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management with participation of the IEEE Communications Society CNOM and with support from the Institute for Educational Services*, pages 697–708, 1993.

Christophe Dousson. Alarm driven supervision for telecommunication network: Ii - on-line chronicle recognition. *Annals of telecommunications*, 51:501–508, 1996.

Christophe Dousson and Thang Vu Duong. Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *Proceedings of the 16th international joint conference on Artifical intelligence - Volume 1*, IJCAI 1999, pages 620–626, 1999.

Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 324–329, 2007.

Alexander Dupuy, Er Dupuy, Soumitra Sengupta, Ouri Wolfson, and Yechiam Yemini. Design of the netmate network management system. In *Zimmer (Eds.), Integrated Network Management II, North-Holland*, pages 639–650, 1991.

David Durham, Jim Boyle, Ron Cohen, Shai Herzog, Raju Rajan, and Arun Sastry. RFC 2748: The COPS (Common Open Policy Service) protocol, 2000.

Eric Fabre. Bayesian Networks of Dynamic Systems. Technical report, Habilitation Thesis in Computer Science, University of Rennes 1, June 2007.

Eric Fabre and Christoforos Hadjicostis. A Trellis Notion for Distributed System Diagnosis with Sequential Semantics. In *8th International Workshop on Discrete Events Systems*, pages 294–300, 2006.

Eric Fabre, Albert Benveniste, Stefan Haar, Claude. Jard, and Armen Aghasaryan. Algorithms for Distributed Fault Management in Telecommunications Networks. In *Telecommunications and Networking*, volume 3124 of *ICT 2004*. Springer Berlin Heidelberg, 2004.

Eric Fabre, Albert Benveniste, Stefan Haar, and Claude Jard. Distributed Monitoring of Concurrent and Asynchronous Systems. *Discrete Event Dynamic Systems*, 15: 33–84, 2005.

Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From Data Mining to Knowledge Discovery: An Overview. In *Advances in Knowledge Discovery and Data Mining*, 1996.

Françoise Fessant, Fabrice Clérot, and Christophe Dousson. Mining of an Alarm Log to Improve the Discovery of Frequent Patterns. In *Advances in Data Mining*, pages 144–152. Springer Berlin Heidelberg, 2005.

Pierre Francois and Olivier Bonaventure. Avoiding transient loops during IGP convergence in IP networks. In *24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1 of *INFOCOM 2005*, pages 237–247, 2005.

Pierre Francois and Olivier Bonaventure. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Transactions on Networking*, 15: 1280–1292, 2007.

Peter Frohlich, Wolfgang Nejdl, Klaus Jobmann, and Hermann Wietgrefe. Model-based alarm correlation in cellular phone networks. In *Proceedings of the 5th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, MASCOTS 1997, pages 197–204, 1997.

Robert D Gardner and David A Harle. Methods and systems for alarm correlation. In *Global Telecommunications Conference. Communications: The Key to Global Prosperity*, GLOBECOM 1996, 1996.

Robert D Gardner and David A Harle. Alarm correlation and network fault resolution using the kohonen self-organising map. In *Global Telecommunications Conference*, GLOBECOM 1997, pages 1398–1402, 1997.

Robert D Gardner and David A Harle. Pattern discovery and specification techniques for alarm correlation. In *Network Operations and Management Symposium*, NOMS 1998, pages 713–722, 1998.

Samir Ghamri-Doudane and Laurent Ciavaglia. Domain-wide Scheduling of OSPF Graceful Restarts for Maintenance Purposes. In *International Conference on Network and Service Management (CNSM)*, 2010.

Minas Gjoka, Vinayak Ram, and Xiaowei Yang. Evaluation of IP Fast Reroute Proposals. In *Proceedings of IEEE COMSWARE '07*, 2007.

Ashok Goel, J Ramanujam, and P Sadayappan. Towards a 'neural' architecture for abductive reasoning. In *IEEE International Conference on Neural Networks*, pages 681–688, 1998.

Shri K Goyal. Knowledge technologies for evolving networks. *Integrated Network Management, II*, pages 439–461, 1991.

I. Grosclaude. Une approche a base de modeles pour les services de depannage (Model-based problem solving for help and support services). In *French Conference on Artificial Intelligence*, RFIA 2008, 2008.

Boris Gruschke et al. Integrated event management: Event correlation using dependency graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management*, DSOM 1998, 1998.

Bruno Guerraz and Christophe Dousson. Chronicles construction starting from the fault model of the system to diagnose. In *15 th International Workshop on Principles of Diagnosis*, DX 2004, pages 51–56, 2004.

Masum Hasan, Binay Sugla, and Ramesh Viswanathan. A conceptual framework for network management event correlation and filtering systems. In *Integrated Network Management*, pages 233–246, 1999.

Kimmo Hätönen, Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, and Hannu Toivonen. Knowledge discovery from telecommunication network alarm databases. In *Proceedings of the Twelfth International Conference on Data Engineering*, ICDE 1996, pages 115–122, 1996.

David Hock, Matthias Hartmann, Tim Neubert, and Michael Menth. Loop-free convergence using ordered FIB updates: Analysis and routing optimization. In *8th International Workshop on the Design of Reliable Communication Networks*, DRCN 2011, pages 156–163, 2011.

K. Houck, S. Calo, and A. Finkel. Towards a practical alarm correlation system. In *Proceedings of the fourth international symposium on Integrated network management IV*, pages 226–237, 1995.

Carole Hounkonnou and Eric Fabre. Empowering Self-diagnosis with Self-modeling. In *8th international conference on Network and Service Management*, CNSM 2012, pages 364–370. IEEE, 22–26 October 2012.

Carole Hounkonnou and Eric Fabre. Enhanced OSPF Graceful Restart. In *IFIP/IEEE International Symposium on Integrated Network Management - TechSessions*, IM 2013, 27-31 May 2013.

Carole Hounkonnou, Samir Ghamri-Doudane, and Eric Fabre. Détection et correction de boucles de routage dans un réseau de routeurs utilisant un protocole de routage de type SPF. Europeean Patent, 2012.

R. Hülsermann, S. Bodamer, M. Barry, A. Betker, M. Jäger, J. Späth, C.M. Gauger, and M. Köhn. A Set of Typical Transport Network Scenarios for Network Modelling. In *Beiträge zur 5. ITG-Fachtagung Photonische Netze*, pages 65–72, May 2004.

Rolf Isermann. Supervision, fault-detection and fault-diagnosis methods — an introduction. *Control Engineering Practice*, 5:639–652, 1997.

Rolf Isermann. Model-based fault-detection and diagnosis — status and applications. *Annual Reviews in Control*, 29:17–85, 2005.

ITU-T. Gateway control protocol: Version 3, ITU-T Recommendation H. 248.1, 2013.

Gabriel Jakobson and Mark Weissman. Real-time telecommunication network management: extending event correlation with temporal constraints. In *Proceedings of the fourth international symposium on Integrated network management IV*, pages 290–301, 1995.

Juniper. Configuring Graceful Restart for OSPF, 2013. URL `http://juniper.net/techpubs/en_US/junos/topics/topic-map/ospf-graceful-restart.html`.

S. Kätker and M. Paterok. Fault isolation and event correlation for integrated fault management. In *Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management V : integrated management in a virtual world: integrated management in a virtual world*, pages 583–596, 1997.

Stefan Katker and Kurt Geihs. A Generic Model for Fault Isolation in Integrated Management Systems. *Journal of Network and Systems Management*, 5:109–130, 1997.

Soila P Kavulya, Kaustubh Joshi, Matti Hiltunen, Scott Daniels, Rajeev Gandhi, and Priya Narasimhan. Draco: Top-down statistical diagnosis of large-scale voip networks, 2011.

Soila P. Kavulya, Kaustubh Joshi, Matti Hiltunen, Scott Daniels, Rajeev Gandhi, and Priya Narasimhan. Practical experiences with chronics discovery in large telecommunications systems. *ACM SIGOPS Operating Systems Review*, 45:23–30, 2012.

Walter Kehl, Heiner Hopfmüller, Traytcho Koussev, and Mark Newstead. Application of model-based reasoning to the maintenance of telecommunication networks. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 79–88. Springer Berlin Heidelberg, 1992.

Rana M Khanafer, Beatriz Solana, Jordi Triola, Raquel Barco, Lars Moltsen, Zwi Altman, and Pedro Lazaro. Automated diagnosis for UMTS networks using Bayesian network approach. *IEEE Transactions on Vehicular Technology*, 57:2451–2461, 2008.

Shmuel Kliger, Shaula Yemini, Yechiam Yemini, David Ohsie, and Salvatore Stolfo. A coding approach to event correlation. In *Proceedings of the fourth international symposium on Integrated network management IV*, pages 266–277, 1995.

Mark L Krieg. A tutorial on Bayesian belief networks. Technical report, DTIC Document, 2001.

Lundy Lewis. A Case-Based Reasoning Approach to the Resolution of Faults in Communication Networks. In *Integrated Network Management*, pages 671–682, 1993.

Lundy Lewis and Gabi Dreo. Extending trouble ticket systems to fault diagnostics. *Network, IEEE*, 7:44–51, 1993.

Dennis V Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27:986–1005, 1956.

Kar-Wing Edward Lor. A Network Diagnostic Expert System for Acculink Multiplexers Based on a General Network Diagnostic Scheme. In *Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management with participation of the IEEE Communications Society CNOM and with support from the Institute for Educational Services*, pages 659–669, 1993.

Jingxian Lu, Christophe Dousson, Benoit Radier, and Francine Krief. Towards an Autonomic Network Architecture for Self-healing in Telecommunications Networks. In *Mechanisms for Autonomous Management of Networks and Services*, volume 6155, pages 110–113. Springer Berlin Heidelberg, 2010.

Jingxian Lu, Christophe Dousson, and Francine Krief. A Self-diagnosis Algorithm Based on Causal Graphs. In *The Seventh International Conference on Autonomic and Autonomous Systems*, ICAS 2011, 2011.

Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 210–215, 1995.

Robert Mathonet, Herwig Van Cotthem, and Leon Vanryckeghem. DANTES: An Expert System for Real-Time Network Troubleshooting. In *Proceedings of the 10th international joint conference on Artificial intelligence - Volume 1*, pages 527–530, 1987.

Cristina Melchiors and Liane M. Rockenbach Tarouco. Fault Management in Computer Networks Using Case-Based Reasoning: DUMBO System. In *Proceedings of the Third International Conference on Case-Based Reasoning and Development*, pages 510–524, 1999.

Masanori Miyazawa and Kosuke Nishimura. Scalable root cause analysis assisted by classified alarm information model based algorithm. In *7th International Conference on Network and Service Management*, CNSM 2011, pages 1–4, 2011.

J. Moy, P. Pillay-Esnault, and A. Lindem. RFC 3623: Graceful OSPF restart, November 2003.

John Moy. *OSPF: Anatomy of an Internet Routing Protocol.* Addison-Wesley Longman Publishing Co., Inc., 1998a.

John Moy. RFC 2328: OSPF Version 2, April 1998b.

Kevin Murphy et al. The Bayes Net Toolbox for Matlab. *Computing science and statistics*, 33:1024–1034, 2001.

RJ Patton, J Chen, and TM Siew. Fault diagnosis in nonlinear dynamic systems via neural networks. In *International Conference on Control*, volume 2, pages 1346–1351, 1994.

L.F. PAU. Survey of expert systems for fault detection, test generation and maintenance. *Expert Systems*, 3:100–110, 1986.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann, 1988.

Georgia C. Penido, José Marcos S. Nogueira, and Christiano Mata Machado. An automatic fault diagnosis and correction system for telecommunications management. In *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 777–791, 1999.

Duc Truong Pham. *Expert Systems in Engineering.* Springer-Verlag New York, Inc., 1988.

Gianluca Reali and Luca Monacelli. Definition and performance evaluation of a fault localization technique for an NGN IMS network. *IEEE Transactions on Network and Service Management*, 6(2):122–136, 2009.

J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: Session initiation protocol, 2002.

Raguram Sasisekharan, Yung-Kao Hsu, and David Simen. SCOUT: an approach to automating diagnoses of faults in large scale networks. In *Global Telecommunications Conference*, GLOBECOM 1993, pages 212–216, 1993.

Raguram Sasisekharan, V Seshadri, and Sholom M Weiss. Data mining and forecasting in large-scale telecommunication networks. *IEEE Expert*, 11:37–43, 1996.

William T Scherer and CC White. A survey of expert systems for equipment maintenance and diagnostics. In *Fault detection & reliability: knowledge based & other approaches*, pages 3–18. Pergamon Press, 1987.

H Schulzrinne, S Casner, R Frederick, and V Jacobson. RFC 3550: RTP: A Transport Protocol for Real-Time Applications, 2003.

Aman Shaikh, Rohit Dube, and Anujan Varma. Avoiding instability during graceful shutdown of OSPF. In *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2 of *INFOCOM 2002*, pages 883–892, 2002.

Aman Shaikh, Rohit Dube, and Anujan Varma. Avoiding instability during graceful shutdown of multiple OSPF routers. *IEEE/ACM Transactions on Networking*, 14 (3):532 –542, 2006.

Stephen Slade. Case-based reasoning: A research paradigm. *AI magazine*, 12:42, 1991.

Malgorzata Steinder and Adarshpal S. Sethi. End-to-end Service Failure Diagnosis Using Belief Networks. In *Network Operations and Management Symposium*, NOMS 2002, pages 375–390, 2002.

Malgorzata Steinder and Adarshpal S. Sethi. A survey of fault localization techniques in computer networks. *Science of Computer Programming*, 53:165–194, 2004a.

Malgorzata Steinder and Adarshpal S. Sethi. Probabilistic fault diagnosis in communication systems through incremental hypothesis updating. *Computer Networks*, 45: 537–562, 2004b.

Yongning Tang, Ehab Al-Shaer, and Raouf Boutaba. Efficient fault diagnosis using incremental alarm correlation and active investigation for internet and overlay networks. *Network and Service Management, IEEE Transactions on*, 5:36–49, 2008.

ETSI TISPAN. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN) - Resource and Admission Control Sub-system (RACS) - Functional Architecture. ETSI ES 282 003 V1.1.1 (2006-06), June 2006.

ETSI TISPAN. Universal Mobile Telecommunications System (UMTS) - Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN) - IP Multimedia Subsystem (IMS) - Functional architecture. ETSI TS 123-517 V8.0.0 (2007-12), December 2007.

ETSI TISPAN. Universal Mobile Telecommunications System (UMTS) - TISPAN - IP Multimedia Subsystem (IMS) - Stage 2 description. ETSI TS 123 506 V8.1.0 (2008-10), October 2008a.

ETSI TISPAN. Universal Mobile Telecommunications System (UMTS) - TISPAN - IP Multimedia Subsystem (IMS) - Stage 2 description. ETSI TS 123 406 V7.2.0 (2008-10), October 2008b.

ETSI TISPAN. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN) - NGN Functional Architecture. ETSI ES 282 001 V3.4.1 (2009-09), September 2009.

ETSI TISPAN. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN) - NGN Functional Architecture - Network Attachment Sub-System (NASS). ETSI ES 282 004 V3.4.1 (2010-03), March 2010a.

ETSI TISPAN. Universal Mobile Telecommunications System (UMTS) - LTE - IP Multimedia Subsystem (IMS) - Stage 2. ETSI TS 123 228 V9.4.0 (2010-10), October 2010b.

ETSI TISPAN. Universal Mobile Telecommunications System (UMTS) - LTE - 3G security - Access security for IP-based services. ETSI TS 133 203 V10.2.0 (2010-12), December 2010c.

Andrew J Weiner, David A Thurman, and Christine M Mitchell. Applying case-based reasoning to aid fault management in supervisory control. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4213–4218, 1995.

Gary Weiss, John Eddy, and Sholom Weiss. Intelligent telecommunication technologies, 1998a.

Gary Weiss, Johannes P. Ros, and Anoop Singhal. ANSWER: Network Monitoring Using Object-Oriented Rules. In *Proceedings of the Tenth Conference on Innovative Applications of Artificial Intelligence*, pages 1087–1093, 1998b.

Sholom M. Weiss and Nitin Indurkhya. Optimized Rule Induction. *IEEE Expert: Intelligent Systems and Their Applications*, 8:61–69, 1993.

Hermann Wietgrefe. Investigation and practical assessment of alarm correlation methods for the use in GSM access networks. In *IEEE/IFIP Network Operations and Management Symposium*, NOMS 2002, pages 391–403, 2002.

Peng Wu, R. Bhatnagar, L. Epshtein, M. Bhandaru, and Zhongwen Shi. Alarm correlation engine (ACE). In *IEEE/IFIP Network Operations and Management Symposium*, NOMS 98, pages 733–742, 1998.

Jie Xiao, Changcheng Huang, and James Yan. IMS network deployment cost optimization based on flow-based traffic model. In *IEEE/IFIP Network Operations and Management Symposium*, NOMS 2010, pages 232–239, 2010.

Shaula Alexander Yemini, Shmuel Kliger, Eyal Mozes, Yechiam Yemini, and David Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, 34:82–90, 1996.

List of Figures

1.1 Les réseaux IMS sont organisés de façon hiérarchique. 6
1.2 Une instance de réseau IMS: la couche physique et les fonctions de la couche fonctionnelle hébergées sont illustrées. 8
1.3 Diagramme de séquence de la configuration IP en utilisant le protocole DHCP (couche procédurale). Cette configuration se traduit par un dialogue entre l'UE (à gauche) et les fonctions du NASS (à droite). 10
1.4 Réseau Bayésien générique pour le service de configuration IP d'un terminal utilisateur. 11
1.5 L'instance du réseau Bayésien relative à Laurie 11
1.6 L'instance de Jane est ajoutée; ses ressources privées sont représentées en bleu. 12
1.7 L'instance du réseau Bayésien relative à Alice est ajoutée; ses ressources privées sont représentées en vert. Les ressources privées de Jane ne sont pas montrées. 13
1.8 Un réseau Bayésien générique (à gauche) avec un pattern unique, et une instance de ce réseau (à droite) avec six instances de ce pattern se chevauchant. 14
1.9 Une instance de réseau Bayésien générique avec 22 instances de pattern. L'épaisseur de la ligne reliant deux instances reflète le nombre de variables partagées. Pour estimer le pattern central X_1, les observations sont introduites par zones, en commençant par la zone la plus foncée et en poursuivant vers les zones plus pales. 16
1.10 Evolution en moyenne de l'entropie conditionnelle du pattern central X_1 au fur et à mesure que le nombre d'observations augmente. 17

3.1 An example of OSPF network 28
3.2 Shortest paths from C to all destinations (the source graph of C). . . . 29
3.3 Destination graphs to F before and after failure of link $D \to F$. 30
3.4 The expected source graph G^{h_k} of h_k, and its misbehavior for packets addressed to δ. Instead of correctly forwarding such packets to u, node r selects a wrong neighbor and actually send them back to h_k, thus creating a circuit. 33
3.5 Vertex colors on the destination graph G_F: green=0, yellow=1, red=2. . 35
3.6 Successive reroutings to correct destination graph G_F. 35
3.7 Successive reroutings to correct destination graph G_F. 36

3.8 Successive removals of the temporary reroutings into the corrected destination graph G'_F, after node C returns to function. 37
3.9 Rerouting packets addressed to δ_1 with a minimal number of hops in order to go around r (and thus avoid the circuit). This rerouting path can also be (partly) used to correct the circuit on the path to δ_2 when $\delta_1 \rightsquigarrow \delta_2$ in G^{H_k}. 39
3.10 Evaluation on the NSFNET network 40

4.1 Organization of rule-based expert systems 47
4.2 An example of forward chaining 48
4.3 Structure of the knowledge sources 52
4.4 Case-Based Reasoning Cycle 55
4.5 A simple data set with two classes used for illustrative purposes. 58
4.6 A simple linear classification boundary for the loan data set. 58
4.7 A Simple Linear Regression for the Loan Data Set. 59
4.8 A Simple Clustering of the Loan Data Set into Three Clusters. 59
4.9 An Example of Classification Boundaries Learned by a Nonlinear Classifier (Such as a Neural Network) for the Loan Data Set. 61
4.10 Model of a feedforward neural network. 63
4.11 A problem-symptom graph and its associated codebook. 64
4.12 A causal graph explaining the propagation of failures from the problems (top) to the symptoms (bottom). This causal graph yields the problem-symptom graph of Figure 4.11 by transitivity closure, after removal or intermediate nodes and duplicated symptoms. 66
4.13 A causal graph and a configuration over it. Edges describe two propagation modalities: "must cause" (solid arrows) and "may cause" (dashed arows). Test nodes are represented as parallelograms, alarm nodes as ovals. 69
4.14 Two causal graphs, for resp. domains 1 and 2, sharing some nodes (p_2, p_3 and a_2). Left: two compatible local configurations. Right: two incompatible local confidurations. 71
4.15 An timed event flow (top) and a chronicle (bottom). The arrows indicate a chronicle instance within the event flow. 74
4.16 The constraint propagation rule in a chronicle. 75
4.17 A time constraint graph before (left) and after (right) constraint propagation. 75
4.18 Merging 3 chronicles of size 2 into a chronicle of size 3, with time constraints propagation. 77
4.19 A Bayesian network for IP paths diagnosis 78
4.20 Propagation in a chain network 81
4.21 Propagation in a tree 81
4.22 Propagation in a causal polytree network 82
4.23 NBM 85

5.1 The generic IMS physical architecture 90
5.2 Core IP Multimedia Subsystem architecture 91
5.3 Session Control 92
5.4 The media functions 94
5.5 Resource and Admission Control Subsystem architecture 96
5.6 Network components and control flow for QoS 97
5.7 Network Attachment Subsystem architecture 98
5.8 Network components and procedure for authentication 99
5.9 TISPAN NGN IMS functional architecture 100
5.10 Mapping functions to physical equipment 101
5.11 Network attachment procedure using DHCP 103
5.12 Notification sent to the RACS during network attachment 104
5.13 IMS-level registration procedure 106
5.14 Call origination procedure 109
5.15 IMS networks are hierarchically organized structures 111

6.1 The generic IMS physical architecture and its instance 114
6.2 The generic IMS functional architecture and its instance 116
6.3 Different levels of granularity for IP configuration procedure 119
6.4 A mapping between the physical and the functional architecture 120
6.5 Network model class hierarchy 121
6.6 Class UE inherits from class Function 122
6.7 An IMS network instance 123

7.1 An IMS network instance 127
7.2 Sequence diagram of IP configuration using DHCP (procedural layer) . 127
7.3 Generic BN for the IP configuration capability of a User Equipment. . . 128
7.4 Laurie's BN instance 129
7.5 Jane's BN instance is added; her private resources are represented in blue 130
7.6 Alice's BN instance is added. 130
7.7 A GBN (left) with a single pattern, and an instance of this GBN (right). 132

8.1 A GBN (left) with a single pattern, and an instance of this GBN (right). 140
8.2 A GBN instance with 22 pattern instances. 140
8.3 Average evolution of the conditional entropy of the central pattern X_1. . 141
8.4 A Network instance example. 142
8.5 IP connectivity sequence diagram. 142
8.6 Pattern for IP connectivity 143
8.7 Mary's pattern instance. 143
8.8 The whole model. 145
8.9 Evolution of H(Z) while pattern instances are added 148
8.10 Average evolution of the conditional entropy of Z. 149
8.11 Examples of non-decreasing behavior of H(Z) 149

Glossary

4G 4th Generation 22

A-RACF Access Resource and Admission Control Function 96

ADSL Asymmetric Digital Subscriber Line 102

AF Application Function 96

AMF Access Management Function 98

AN Access Node 89

ARF Access Relay Function 98

AS Application Server 95

BGCF Breakout Gateway Control Function 94

BGF Border Gateway Function 96

BRAS Broadband Remote Access Server 90

C-BGF Core Border Gateway Function 96

CLF Connectivity Session Location Function 98

CNG Customer Network Gateway 99

CNGCF Customer Network Gateway Configuration Function 99

COPS Common Open Policy Service 91

CPE Customer Premise Equipment 89

CS Circuit Switched 94

CSCF Call Session Control Function 91

DHCP Dynamic Host Configuration Protocol 98, 102

DSL Digital Subscriber Line 90

DSLAM DSL Access Multiplexer (DSLAM) 89

DSP Digital Signal Processing 94

FQDN Fully Qualified Domain Name 104

FTTH Fiber-to-the-home 21

I-CSCF Interrogating CSCF 91

IBCF Interconnection Border Control Function 95

IMS IP Multimedia Subsytem 2, 22, 25, 89

IMS-MGW IMS Media Gateway 94

IP Internet Protocol 21, 90

IP-CAN IP Connectivity Access Network 102

ISC IMS Service Control 95

ISUP ISDN User Part 94

IWF Inter Working Function 95

MGCF Media Gateway Control Function 94

MRF Multimedia Resource Function 93

MRFC Multimedia Resource Function Controller 94

MRFP Multimedia Resource Function Processor 94

NACF Network Access Configuration Function 98

NAPT Network Address Port Translation 96

NASS Network Attachment Subsystem 91

NAT Network Address Translation 96

NGN Next Generation Network 22, 104

OSPF Open Shortest Path First 24

P-CSCF Proxy CSCF 91

PDBF Profile Database Function 98

PoP Point of Presence 90

PPP Point-to-Point Protocol 102

QoS Quality of Service 95

RACS Resource and Admission Control Subsystem 91

RCEF Resource Control Enforcement Function 96

RTP Real-Time Protocol 91

S-CSCF Serving CSCF 91

SBC Session Border Controller 90

SDP Session Description Protocol 93

SGW Signaling Gateway 95

SIP Session Initiation Protocol 90

SLF Subscriber Locator Function 93

SPDF Service Policy Decision Function 96

TDM Time Division Multiplexing 94

UAAF User Access Authorization Function 98

UE User Equipment 98

UMTS Universal Mobile Telecommunications Services 22

UPSF User Profile Server Function 91

URI Uniform Resource Identifier 105

VU :

Le Directeur de Thèse
(Nom et Prénom)
FABRE Eric

VU :

Le Responsable de l'École Doctorale

J-N LION

ECOLE DOCTORALE MATISSE
Le Directeur

DT UR1 2013/3/7 n°32

VU pour autorisation de soutenance

Rennes, le 3 juillet 2013

Le Président de l'Université de Rennes 1

Guy CATHELINEAU

Le Président et par délégation
le Vice-Président

VU après soutenance pour autorisation de publication :

Le Président de Jury,
(Nom et Prénom)

LAGRANGE Xavier