Diss. ETH No. 26177

Qualitative and Quantitative Guarantees for Access Control

A thesis submitted to attain the degree of

Doctor of Sciences of ETH ZURICH

presented by

THILO WEGHORN

Diplom-Mathematiker Univ.
Diplom-Informatiker Univ.
LMU Munich

born on 15.10.1985

citizen of Germany

accepted on the recommendation of

Prof. Dr. David Basin, examiner
Prof. Dr. Adrian Perrig, co-examiner
Prof. Dr. David Hausheer, co-examiner

2019

Abstract

Since life began spreading on Earth, the distribution of vital resources and their subsequent safekeeping have led to constant survival conflicts. Even more so in today's technologically and economically advanced societies, where wrongly granted access to critical resources, like private customer data, powerful weaponry, or crucial infrastructure can harm organizations, civilizations, or even life itself.

A famous example where poorly managed access led to the fall of a large organization is the case of Barings Bank [1], which went bankrupt after one of its employees engaged in a series of fraudulent and unauthorized investments. Moreover, crucial bandwidth resources in today's Internet suffer from major distribution vulnerabilities. Distributed Denial of Service (DDoS) attacks against GitHub, in the beginning of 2018, peaking at 1.35 Tbps, and against Arbor Networks, only five days later, peaking at 1.7 Tbps [64], show that attackers can severely skew bandwidth allocations to harm large organizations. Verizon's data breach report from 2019 asserts that denial of service and privilege misuse are the two major causes for security incidents we face today [2].

To address these two problems, sophisticated techniques were invented in the following two fields. (1) *Access control* provides **qualitative guarantees** by defining and enforcing access control policies. These policies are designed to give honest users access to crucial resources and to prevent malicious users from nefarious actions. *Policy analysis* applies logic-based frameworks to specify and verify properties of such policies. (2) *Resource allocation* provides **quantitative guarantees** by fairly distributing resources among users. *Bandwidth reservation* provides hard bandwidth and stability guarantees in the face of strong network adversaries during DDoS attacks.

This thesis makes following contributions to these two fields. First, we propose **FORBAC**, an extension of Role-Based Access Control (RBAC) based on first-order logic. FORBAC is expressive enough to formalize a wide range of access control policies and is simple enough to keep their policy analysis in NP. Second, we provide the first principled solution to the global bandwidth allocation problem in adversarial networks. We propose and formalize the **N-Tube** algorithm, that reserves bandwidth in path-aware networks with a powerful attacker. We prove that N-Tube's bandwidth allocations quickly stabilize, provide minimum bandwidth guarantees, even in the presence of congestion, and satisfy a new fairness notion.

Zusammenfassung

Seitdem sich Leben auf der Erde auszubreiten begann, haben die Verteilung von lebenswichtigen Ressourcen und deren anschließende Sicherstellung ständig zu Überlebenskonflikten geführt. Dies gilt heutzutage umso mehr in technologisch und wirtschaftlich fortschrittlichen Gesellschaften, in denen falsch vergebener Zugriff auf kritische Ressourcen, wie private Kundendaten, mächtige Waffensysteme oder lebenswichtige Infrastrukturen, erheblichen Schaden für Organisationen, Zivilisationen und sogar das Leben an sich verursachen kann.

Ein berühmtes Beispiel, bei dem schlecht verwaltete Zugangskontrollen zum Sturz eines großen Unternehmens führten, ist der Fall der Barings Bank, die bankrottging, nachdem einer ihrer Mitarbeiter eine Reihe von betrügerischen und ungenehmigten Investitionen getätigt hatte. Darüber hinaus leiden wichtige Bandbreitenressourcen im heutigen Internet unter schwerwiegenden Verteilungsschwächen. Verteilte Denial of Service (DDoS)-Attacken gegen GitHub Anfang 2018 mit Datenraten von bis zu 1,35 Tbit/s pro Sekunde und nur fünf Tage später gegen Arbor Networks mit bis zu 1,7 Tbit/s, zeigen, dass Angreifer Bandbreitenzuweisungen erheblich verdrehen können, um großen Unternehmen Schaden zuzufügen. Verizons Bericht zu Datenschutzverletzungen aus dem Jahr 2019 besagt, dass Denial-of-Service und Missbrauch von Zugriffsrechten die beiden Hauptursachen für heutige Sicherheitsvorfälle sind [2].

Um diese beiden Probleme anzugehen, wurden ausgefeilte Techniken in den folgenden beiden Bereichen entwickelt: (1) Der Bereich der *Zugriffskontrolle* bietet **qualitative Garantien**, indem mit Hilfe von Zugriffskontrollrichtlinien gutartigen Benutzern der Zugriff auf wichtige Ressourcen gewährt wird und böswillige Benutzer von schändlichen Handlungen abgehalten werden. Die *Richtlinienanalyse* verwendet logikbasierte Modelle um Eigenschaften von Zugriffskontrollrichtlinien zu definieren und zu verfizieren. (2) Der Bereich der *Ressourcenzuteilung* bietet **quantitative Garantien**, indem die Ressourcen fair unter verteilt werden. *Bandbreitenreservierungen* bieten bei DDoS-Attacken, die durch mächtige Netzwerksangreifer verursacht werden, hohe Bandbreiten- und Stabilitätsgarantien.

Diese Arbeit liefert zu den beiden Bereichen die folgenden Beiträge: Zunächst präsentieren wir **FORBAC**, eine Erweiterung der rollenbasierten Zugriffssteuerung (RBAC), welche auf Prädikatenlogik basiert. FORBAC ist aussagekräftig genug, um eine breite Palette von Zugriffssteuerungsrichtlinien zu formalisieren, und ist einfach genug, um ihre Richtlinienanalyse auf die Komplexitätsklasse NP zu beschränken. Zweitens bieten wir die erste anhand von Grundsätzen abgeleitete Lösung für das globale Bandbreitenzuweisungsproblem in Netzwerken mit Attackern. Wir präsentieren und formalisieren den **N-Tube**-Algorithmus, der Bandbreitenreservationen in pfadbewussten Netzwerken mit starken Angreifern ermöglicht. Wir beweisen, dass sich die Bandbreitenzuweisungen von N-Tube schnell stabilisieren, selbst während Netzüberlastungen eine *Mindestbandbreite* garantieren und einen neuen Fairnessbegriff erfüllen.

Acknowledgments

I would like to thank my advisor, David Basin, for providing me with the opportunity to study at ETH, working together with industry partners, and doing research in many interesting fields during my doctoral studies. His support, dedication and mentorship helped me greatly on this journey. Especially his fast feedback, often on weekends or late at night, was very inspiring for me. I would also like to thank my co-examiners Adrian Perrig and David Hausheer for taking the time to read my thesis, evaluating the ideas presented, and providing valuable feedback for further improvement.

I would also like to express my gratitude to those who in one way or another have mentored me throughout my time at ETH Zurich. I would like to thank Manuel Clavel for his guidance in the beginning of my PhD as well as our colaborators Claudiu Duma, Thomas Moser, Benjamin Wehe, and Gunnar Schröder at Credit Suisse for providing me a glimpse in the challenges of access control in the banking enterprise setting.

I am particularly thankful to Christoph Sprenger, Esfandiar Mohammadi, Marco Guarnieri, and Petar Tsankov for providing very helpful advice and feedback throughout my doctoral studies, to Stephen Shirley, Christos Pappas, Tae-Ho Lee, Dominik Roos, and Milan Pandurov for the fruitful discussions about building a bandwidth reservation algorithm for future Internet architectures, as well as to Barbara Pfänder and Bernadette Gianesi for their great administrative support.

I was very lucky to have had great colleagues like Hubert Ritzdorf, Ognjen Marić, Aritra Dhar, and Der-Yeuan Yu who made my time at ETH such a pleasurable experience and helped to obtain a better understanding of security and computer science in general.

I would like to especially thank Carlos Cotrini, Luka Mališa, Dietrich Aumann [23], and Joel Reardon for the many interesting discussions about deep topics regarding life in general. Their advice helped me countless times in my personal life and let me grow to the person that I am now. Many thanks also to my family and friends, for all their support during my studies.

Finally, my biggest gratitude goes to Coralie Roche for supporting me in the challenging last stages of my doctoral studies in the best way I could have ever wished for.

Contents

1 Introduction **1**
- 1.1 Background 2
- 1.2 Challenges 8
- 1.3 Contributions 9
- 1.4 Organization 11

2 FORBAC : First-order Role-based Access Control Policy Analysis **13**
- 2.1 Introduction 13
- 2.2 Requirements for FORBAC 15
- 2.3 Syntax and semantics of FORBAC 17
- 2.4 Policy analysis in FORBAC 23
- 2.5 Experimental results 31
- 2.6 Related work 37
- 2.7 Conclusion and future work 40

3 N-Tube: Neighbor-Based Tube-Fair Bandwidth Reservation **43**
- 3.1 Introduction 43
- 3.2 Background 45
- 3.3 N-Tube overview 47
- 3.4 Algorithm details 52
- 3.5 Model 64
- 3.6 Properties 70
- 3.7 Related work 74
- 3.8 Conclusion and future work 76

4 Conclusion **77**
- 4.1 Summary 77
- 4.2 Future Work 78

Appendices **81**

A Appendix: Model **81**
- A.1 Network and Environment 81
- A.2 Paths 82
- A.3 Messages 83
- A.4 Reservation Maps 85
- A.5 Events 86
- A.6 Event Guards 87

A.7 Message-Creation 93
A.8 Reservation Message Arrival Events 95
A.9 N-Tube's Bandwidth Allocation Computation 100
A.10 Deletion Events 107
A.11 Drop Events 109
A.12 Attack Events 109
A.13 Executions 110

B Appendix: Proofs 115
B.1 Event Lemmata 115
B.2 Attacker Lemmata 121
B.3 In Buffer Lemmata 123
B.4 Step Lemmata 131
B.5 Uniqueness Lemmata 135
B.6 Reserved Lemmata 140
B.7 Prefix Lemmata 151
B.8 Theorem 1 158
B.9 Theorem 2 160
B.10 Properties of N-Tube's Bandwidth Allocation Computation . 161

Bibliography 181

Resume 187

List of Figures

1.1 The access control policy *Auth* is given as a relation between users and permissions. 3
1.2 The access control policy *Auth* given in RBAC. 4
1.3 The access control policy *Auth* given in RBAC with constraints. 5
1.4 A two-link network with three users and three paths. . . . 6

2.1 The role r_2 is redundant: the permissions assigned to r_2 are contained in those assigned to r_1 and the users assigned to r_2 are also assigned to r_1. 14
2.2 An example of a Σ_B-structure 19
2.3 User and permission-assignments in the Σ_B-structure S . . . 22
2.4 User and permission-assignments in the Σ_B-structure $\tilde{S}$ . . . 26

3.1 The process of making a reservation. 48
3.2 N-Tube's bandwidth allocation computation at AS_3 distributes the egress link's (adjusted) bandwidth capacity D proportionally to three ingress demands. 49
3.3 Limit demands on the ingress interface. 50
3.4 Limit demands on the egress interface. 51
3.5 Minimum bandwidth guarantee in the worst case. 51
3.6 The message m is processed along p by the events described in Section 3.5.3. 55

Chapter 1
Introduction

One of the earliest conflicts for living beings on our planet has been gaining and defending access to vital resources. The very process of evolution is driven by changes in the environment and their effects on the distribution and accessibility of essential resources like living space, energy supply, and food. In case of resource scarcity, competing life forms have to strike a balance in distributing these resources among each other. This balance usually emerged due to a self-organized distributed process, called natural selection, which is based on better chances of reproduction for the fittest.

Particularly, the human species, one of the currently most successful species regarding adaptation to various environments and the domination of other species, draws a huge part of its success from using its superiority in communication, planning, and sharing resources by trading. Historical evidences of successful but also questionable advances in resource distribution are the Roman Empire which built streets and water supply systems spanning the European continent, the English and Spanish imperialism which extended their resource extraction throughout the world with colonisation, capitalism which optimized production to balance supply and demand of resources, and lately the internet and mobile networks which provides ubiquitous access to information.

However, distributing resources only marked the first step. To avoid losing resources or giving access to resources that might risk the survival of organizations or even states, it became necessary to control their access. Due to technological advances, humans have developed increasingly powerful and dangerous tools that need to be protected from human greed and their urge for power. Wrongly granting malicious users access to such resources has led to privilege misuses and severe damages. Famous examples are the case of Barings Bank [1], which went bankrupt in 1995 after its employee Nick Leeson engaged in a series of fraudulent and unauthorized investments. Another case was the 2011 UBS rogue trader scandal [20] where the trader Kweku Adoboli was convicted of illegally trading away USD 2 billion.

Due to such severe failures in access control, the banking industry has become highly regulated. However, newly emerging technologies like today's Internet are still developing the right means to fairly allocate their resources among users. Access to bandwidth, the main resource in today's Internet, is barely controlled and often volatile. Bandwidth allocation on network links is deployed according to a best-effort service model and therefore vulnerable to traffic overload, also called congestion. Moreover, malicious users can intentionally cause congestion by launching Distributed Denial of Service (DDoS) attacks. To create this artificially induced congestion, attackers control a set of devices connected to the Internet, so called bot-nets. During such an attack these devices collude and generate more traffic than a target can handle or congest critical network links by leaving only a small portion of bandwidth to honest users. DDoS attacks against GitHub, in the beginning of 2018, peaking at 1.35 Tbps [65] and against Arbor Networks, only five days later, peaking at 1.7 Tbps [64], show that allocation of bandwidth resources can be skewed towards extremely unfair allocations in today's Internet. Such allocation imbalances call for new procedures of resource allocation that provide fair and stable access guarantees.

1.1 Background

To regulate access, sophisticated techniques were invented in the fields of **access control** to enable or prevent malicious users to access valuable resources, as well as **resource allocation** to fairly distribute network resources among honest users. In this work, we differentiate between *qualitative* and *quantitative* access guarantees. Given a user u and a resource r, qualitative guarantees ensure that u is given or denied access to r according to a policy. For example, in a bank, relationship managers are allowed to read the balances of their customers' bank accounts but software engineers are denied to make trading transactions. Quantitative guarantees ensure that a certain fraction of r is allocated to u. For example, mobile phone users in the Zurich main station are always guaranteed a minimum amount of bandwidth, or students have stable access to their university's Wi-Fi network during lectures.

Access Control

In this thesis, *qualitative access* is defined by *access control policies*. An *access control policy* is a relation *Auth* between a set of *users* U and a set of *permissions* P. This is illustrated in Figure 1.1, where the access control policy *Auth* is visualized by arrows from users, i.e., employees in an organization, to three permissions: access to servers, printers, and keys.

We say that *user* u *has access to a permission* p, if $(u,p) \in Auth$ and *a user* u *is denied access to a permission* p, if $(u,p) \notin Auth$. Permissions are often further refined to pairs consisting of a *resource* and an *action*, i.e.,

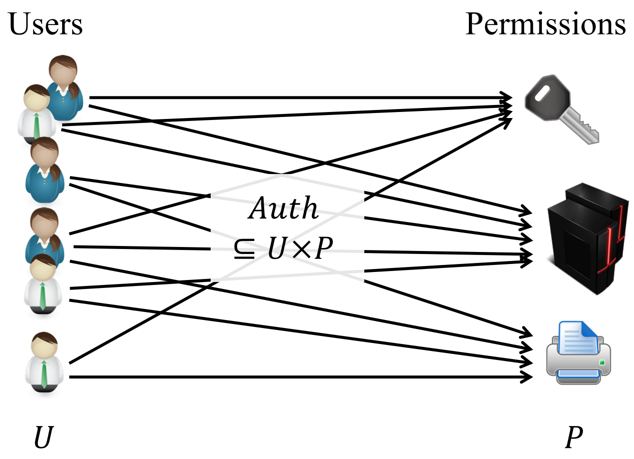


Figure 1.1: The access control policy *Auth* is given as a relation between users and permissions.

$p = (a, r)$. In this case, we say that *a user u is allowed (or denied) to execute an action a on a resource r* .

Access control policies can be given as tables containing all user-permission pairs. However, such policies are mostly defined by procedures, based on high-level languages, that describe when a given users has (or is denied) access to a given permission. To formalize such procedures, many *policy languages* have been invented that can be specified in a machine-readable format and therefore can automatically make access decisions. One of the most commonly used paradigms to specify access control policies is *role-based access control (RBAC)*.

RBAC [26]: To define an access control policy *Auth* in the RBAC paradigm, a third set R and two relations UA and PA are added. Elements of R are called *roles* where each role describes a set of permissions that are needed to execute a certain task, e.g., the role of a relationship-manager who needs read and write permissions to update customer data. The *user-assignment* relation $UA \subseteq U \times R$ assigns roles to users and the *permission-assignment* relation $PA \subseteq R \times P$ assigns permissions to roles. For $(u, r) \in UA$, we say that *a user u has a role r*. Analogously, for $(r, p) \in PA$ we say that *a role r has a permission p*. The *access control policy Auth* is $UA \circ PA$. That is a user u has a permission p, i.e., $(u, p) \in Auth$, if there is a role r such that

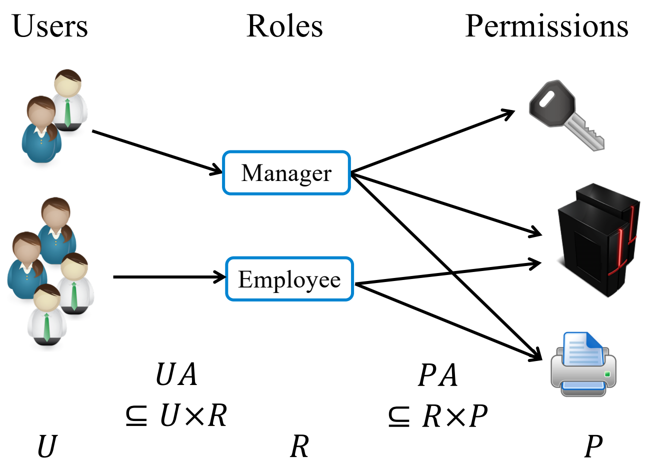


Figure 1.2: The access control policy *Auth* given in RBAC.

$(u, r) \in UA$ and $(r, p) \in PA$. More formally,

$$(u, p) \in Auth \;:\Longleftrightarrow\; \exists r \in R.\ (u, r) \in UA \ \wedge\ (r, p) \in PA.$$

This is illustrated by the example given in Figure 1.2. Employees of an organization are assigned to two roles, manager or employee, given by relation *UA*. The manager-role contains all three permissions, whereas the employee role just provides access to the servers and the printers.

RBAC with constraints [6]: RBAC is not expressive enough for specifying access control policies for large organizations, since it does not scale. To counter this, several extensions of RBAC have been invented. An expressive category of these extensions is called *RBAC with constraints* and allows one to define additional features such as role hierarchies or specifying attributes for users, roles, and permissions. These constraints restrict the relations between user and roles, and roles and permissions, e.g., to be only valid for certain time periods or at specific locations.

In the extended example shown in Figure 1.3 users are partitioned according to their *level*, the manager role contains a set of *goals*, each employee role is connected to a *supervisor*, and the permission granting access to servers has an attribute *ip*. There is a user-assignment constraint that assigns users with level greater than `9` a manager-role with `compliance` and `profit` goals and there is a permission-assignment constraint that assigns a server-permission with IP `1.2.3.4` to employee-roles with supervisor `Bob`.

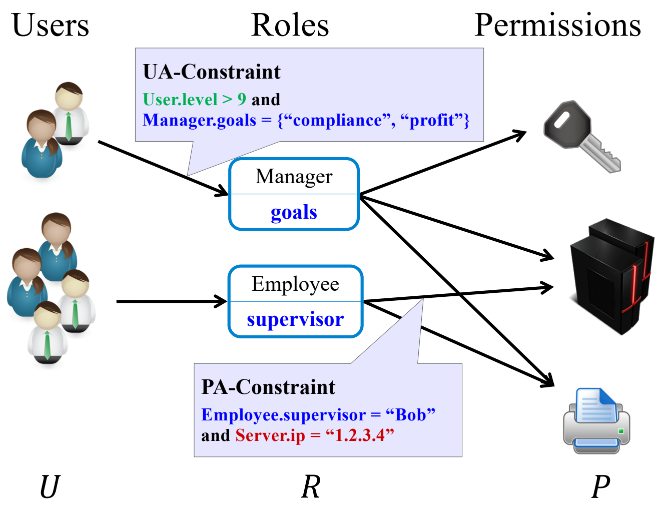


Figure 1.3: The access control policy *Auth* given in RBAC with constraints.

Policy Analysis [28, 54]: Due to the increasing complexity of access control paradigms and their policies the field of *policy analysis* emerged. Policy analysis applies logic-based frameworks to define properties of access control policies and creates methods and tools to verify these properties. These properties are derived from *policy analysis queries* which *policy administrators* must answer to auditors or company management. Common queries are: "Is it possible for all users with a level greater than `10` to access a server with IP `1.2.3.4`?" or "Is there an employee with level less than `5` who has access to the keys?". These queries are then formalized in policy analysis frameworks and verified using policy analysis tools and techniques to provide qualitative access guarantees.

Resource Allocation

A *network* is a weighted, directed graph $N = (V, E, cap)$ with a non-empty set V of nodes, an edge relation $E \subseteq V \times V$, and a weight function $cap : V \to \mathbb{R}_0^+$. Every edge $e \in E$ has a weight $c_e := cap(e)$ assigned to it. A *path* p in this graph is a non-empty sequence of edges of the form $((u, v), (v, w), (w, x), \ldots)$ and its source $src(p)$ is the node u. Nodes in the graph represent *autonomous systems (AS)* in today's Internet, but we refer to them as *users*. Edges represent physical *links* between ASes. Weights represent the links' *bandwidth capacities* and we refer to them as *resources*.

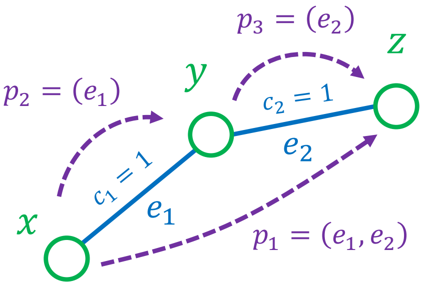


Figure 1.4: A two-link network with three users and three paths.

Figure 1.4 illustrates a network consisting of two links e_1 and e_2 with capacities $c_1 = c_2 = 1$ and three users x,y, and z. The resources are the available bandwidths on the three paths $p_1 = (e_1, e_2)$, $p_2 = (e_1)$, and $p_3 = (e_2)$. User x is the source of p_1 and p_2, and user y is the source of p_3.

Suppose that a network N and a subset of its paths $\mathcal{P}$ are given. Then the *resource allocation problem* consists of allocating a bandwidth x_p to each path $p \in \mathcal{P}$ such that each user $src(p)$ obtains a "fair share". We define fairness next.

Fairness: A *fair* bandwidth allocation is formalized with respect to the utility users derive from the bandwidth allocated to their paths. Suppose that for each user u there exists a numeric function $\mathcal{U}_u$ denoting how much utility u obtains from the network's bandwidth allocations. A solution $\boldsymbol{x}^* = (x_p)_{p \in \mathcal{P}}$ is *fair* if it maximizes the optimization problem

$$\max_{\boldsymbol{x}} \sum_{u \in V} \mathcal{U}_u(\boldsymbol{x})$$

under the constraints

$$\forall e \in E. \sum_{\substack{p \in \mathcal{P} \\ \text{s.t. } e \in p}} x_p \leq c_e$$

$$\forall p \in \mathcal{P}.\ x_p \geq 0.$$

The first constraint states that bandwidth allocations must not exceed the links' capacities and the second constraint ensures that bandwidth allocations are non-negative. The literature refers to this optimization problem as *network utility maximization (NUM)*, because its objective can be understood as to maximize the *social welfare utility* $\sum_{u \in V} \mathcal{U}_u$ given by the aggregated utility of all users.

For the family of utility functions

$$\mathcal{U}_u(\boldsymbol{x}) = \sum_{src(p)=u} \lim_{\alpha \to \infty} -\alpha x_p^{-\alpha},$$

the optimal resource allocation is $\boldsymbol{x}^* = (0.5, 0.5, 0.5)$ and is called *max-min fair*. One can show that

$$\lim_{\alpha \to \infty} -\alpha x_p^{-\alpha} = \begin{cases} -\infty, & \text{if } x_p \in [0;1] \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, this utility function models that users are "infinitely" unsatisfied if they are not allocated a minimum share of 1 and their utility remains constant otherwise. In the max-min fair allocation the allocated bandwidths of smaller or *minimum* flows are *maximized*, i.e., the max-min fair allocation gives maximum protection to the flows with the minimum allocated bandwidths.

For the family of utility functions

$$\mathcal{U}_u(\boldsymbol{x}) = \sum_{src(p)=u} \log(x_p)$$

the optimal resource allocation is $\boldsymbol{x}^* = (1/3, 2/3, 2/3)$ and is called *proportionally fair*. This allocation aims to strike a balance between maximizing the average utilization of all links capacities while at the same time allowing all users a minimum of service.

Resource Allocation [61]: A user, whose utility can be characterized with a concave and differentiable function, is called *elastic*, and *inelastic* otherwise. Under the assumption that all users are elastic, Kelly et al. showed in their seminal paper [45] that the corresponding resource allocation problem has a unique solution. Moreover, they showed that the resource allocation problem can be decomposed into subproblems that can be solved locally at each source and each link in the network. This results in a distributed algorithm that computes the fair resource allocation and provides the basis for different versions of the predominant transport layer protocol in today's Internet, the Transmission Control Protocol (TCP), and various queuing protocols. Using this framework, they proved that these protocols approximate proportional fairness [61].

Bandwidth Reservation: In today's Internet, however, we also find *inelastic* users, i.e., those who require minimum bandwidth guarantees or malicious users whose goal is to reduce other users' utility ad infinitum. These users contradict the utility assumptions given above. Therefore, fair resource allocation is no longer sufficient and *admission control mechanisms* are needed which regulate access to bandwidth resources.

A promising admission control mechanism was proposed for networks supporting path-based forwarding. SIBRA [10], describes a scalable interdomain bandwidth allocation architecture that allows ASes to make and enforce reservations along network paths. The SIBRA architecture is based on a distributed bandwidth reservation algorithm that processes reservation requests made by ASes and allocates bandwidth on the paths' links accordingly. Together with an enforcement mechanism that monitors and polices these reservations, this architecture is designed to provide minimum bandwidth guarantees to all honest users.

1.2 Challenges

We identified four challenges to enable **qualitative** and **quantitative** guarantees. Two major challenges in the field of policy analysis are (i) to define logic-based frameworks that can both express access control policies and their properties, and (ii) to create methods and tools to check these properties efficiently. Two major challenges in the field of bandwidth reservation are (iii) to define distributed algorithms with bandwidth, fairness, and stability guarantees in the face of strong network adversaries, and (iv) to formalize and verify these guarantees. We now elaborate on these challenges for each field.

Imbalance between expressiveness of access control languages and the complexity of their policy analysis: Researchers have investigated numerous extensions that allow RBAC to scale better and to ease its administration, e.g., [29, 34, 40, 42, 48, 52]. However, the expressive power of these extensions makes it difficult to understand the behavior of policies, which in turn has motivated a plethora of research on policy analysis for RBAC, e.g., [6, 11, 27, 62]. Many RBAC extensions use first-order logic in their syntax, but first-order logic is too expressive and needs to be restricted to fragments for enabling efficient policy analysis.

The imbalance between expressiveness and efficient analysis introduces to a new research direction: to develop frameworks expressive enough to formalize realistic access control policies, but simple enough to provide efficient policy analysis in practice. These frameworks should provide languages to specify policies and properties, and methods and tools to verify properties against their policies.

To the best of our knowledge, no prior work has attempted to establish a framework that balances expressiveness and efficient policy analysis for RBAC extensions based on first-order logic.

Verified bandwidth reservation under link-flooding attacks: A core challenge for quantitative bandwidth guarantees is that link-flooding attacks can be caused by a huge number of low-volume flows originating from colluding legitimate-looking bots, e.g., as seen in the Hidden Cobra DDoS Botnet Infrastructure [66]. Therefore, commonly known fairness notions that QoS solutions try to achieve, such as per source [53], per destination [71], per flow [22], per computation [56], and per class [35], are insufficient in such settings and result in an unfair sharing of bandwidth. These fairness notions suffer from the "tragedy of the commons" [32], whereby the incentive of rational users to increase their share of a commonly available resources will lead to infinitesimally small shares for less aggressive, honest users. In particular, in today's Internet, TCP fairness is the most commonly used *per-flow fairness* notion, which allows adversarial users to request arbitrarily many flows and thereby obtain a disproportional amount of bandwidth compared to honest users [18].

To provide bandwidth and stability guarantees several bandwidth reservation architectures have been proposed in the literature, e.g., [10, 25, 35, 72]. These architectures enable users to make bandwidth reservations along network paths by allowing reservation state at the routers. However, they neither handle malicious reservations nor do their authors provide formal arguments to support their claims.

To the best of our knowledge, we are the first to provide a principled solution to the bandwidth allocation problem that provides fairness, minimum bandwidth, and stability guarantees in adversarial networks.

1.3 Contributions

We address the challenges given above with the following two contributions.

FORBAC is an extension of RBAC based on first-order logic. FORBAC is designed to be expressive enough to formalize a wide range of access control policies and at the same time simple enough to provide efficient policy analysis. FORBAC was developed in collaboration with Carlos Cotrini. I proposed relevant policy properties and formalized them in first-order logic. Carlos Cotrini proposed the FORBAC language that allowed to translate the proposed policy properties to existential FORBAC formulas. He also proved that FORBAC is simple enough that these properties can be analyzed in NP and that it is the natural complexity class for this problem. I reduced their policy analysis to the problem of satisfiability modulo appropriate theories. To evaluate FORBAC's expressiveness and our approach to policy analysis, I used off-the-shelf SMT solvers to conducted a case study where I analyzed access control of a major European bank.

N-Tube, a Neighbor-based Tube-fair bandwidth reservation algorithm, allows ASes to reserve bandwidth on network paths. To allocate bandwidth on a path, each AS on the path computes and allocates bandwidth locally while accounting for other reservations. We model the N-Tube algorithm and a powerful attacker as a labeled transition system. Transitions correspond to message exchanges and the creation, update, and deletion of reservations. We formalize and prove that N-Tube's bandwidth allocations *quickly stabilize* in periods of constant demands and that the resulting stable state (i) guarantees the allocation of a *minimum bandwidth*, even in the presence of congestion, and (ii) satisfies a new fairness notion called *bounded tube-fairness*. We also prove that any successful reservation immediately reserves some bandwidth and existing reservations are *immutable* up to their expiration time.

Publications: The content of the two contributions of this thesis are based on the following two articles:

- Carlos Cotrini, Thilo Weghorn, Manuel Clavel, and David Basin, **Analyzing First-Order Role-Based Access Control**, in Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF 2015).
- Thilo Weghorn, David Basin, Adrian Perrig, and Christoph Sprenger, **N-Tube: Guaranteed Bandwidth Reservation in the Age of DDoS**, submitted to CCS 2019.

In addition to the core publications, during my doctoral studies I also co-authored the following articles:

- Carlos Cotrini, Thilo Weghorn, and David Basin, **Mining ABAC Rules from Sparse Logs**, in Proceedings of the 3rd European Symposium on Security and Privacy (EuroS&P 2018).
- Carlos Cotrini, Luca Corinzia, Thilo Weghorn, and David Basin, **The Next 700 Policy Miners: A Universal Method for Building Policy Miners**, submitted to CCS 2019.

Both works are contributions in the field of access control, more specifically in policy mining, closely related to the first part of the thesis.

1.4 Organization

The remainder of this thesis is divided into two parts. First, in Chapter 2 we define our access control policy paradigm FORBAC together with a set of properties given as policy analysis queries in first-order logic. We explain how the queries provide qualitative guarantees for FORBAC-policies and show how we answer them with our tool. Second, in Chapter 3 we model our distributed algorithm N-Tube that reserves bandwidth on network paths and we prove quantitative guarantees such as lower bounds on N-Tube's bandwidth allocations and the time needed for these allocations to stabilize. We conclude the thesis in Chapter 4 by summarizing our findings and providing possible future directions to extend our contributions. Appendices A and B contain technical details and proofs for Chapter 3.

Chapter 2
FORBAC : First-order Role-based Access Control Policy Analysis

2.1 Introduction

RBAC [26] is a predominant access control model for centralized access-control. However, it is not the last word, and researchers have investigated numerous extensions that allow RBAC to scale better and be easier to administrate, e.g. [29, 34, 40, 42, 48, 52]. However, the expressive power of these extensions makes it difficult to understand the behavior of policies, which in turn has motivated a plethora of research on policy analysis for RBAC, e.g. [6, 11, 27, 62].

Many RBAC extensions use first-order logic in their syntax, but first-order logic is simply too expressive for policy specification languages. This is reflected in the syntax of different logic-based languages [12, 31] that have been used in practice; for instance, they exclude disjunction and limit quantifier alternation. Moreover, these languages have been defined with a focus on policy formulation rather than policy analysis. As a result, policy analysis can handle only fragments of these languages. For example, [38] defines a language for administrating user attributes, where first-order logic is used to define administrative rules that specify how users' attribute values change. Later, in [39], the authors study the complexity of the reachability problem, a common analysis problem in administrative RBAC [5, 27, 37]. It turns out that this problem is PSPACE-complete, even after restricting quantifier alternation, and allowing only unary functions and binary predicates. Further restrictions must be imposed on the language to obtain fragments where this reachability problem is solvable in polynomial time.

The use of first-order logic in RBAC extensions gives rise to new problems. In some extensions, the assignments of roles to users and permissions to roles are specified by first-order formulas [14, 29, 34, 40]. This specification is done by humans and is hence prone to errors. Policy administrators may fail to anticipate all the consequences of their specifications. For example, they may specify policies with redundant roles, as illustrated in Figure 2.1, or even worse, assign users incorrect authorizations.

The imbalance between expressiveness and efficient analysis gives rise to a new research direction: to develop frameworks strong enough to express realistic authorization policies, but simple enough to be analyzed

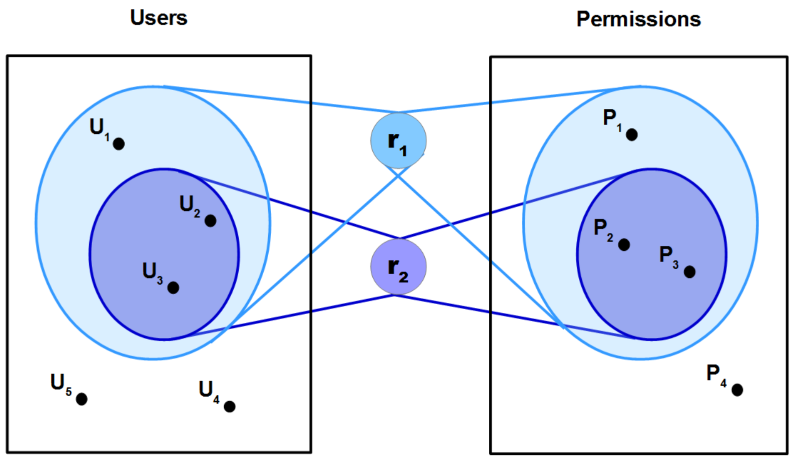


Figure 2.1: The role r_2 is redundant: the permissions assigned to r_2 are contained in those assigned to r_1 and the users assigned to r_2 are also assigned to r_1.

in practice. These frameworks should provide languages for specifying policies and properties, and procedures to verify properties against policies. Other researchers have presented such frameworks [7, 55]. However there are features and problems specific to extensions of RBAC, like the one illustrated in Figure 2.1 [6], that were not addressed by this work. To the best of our knowledge, no prior work has attempted to establish a framework that balances expressiveness and efficient policy analysis for RBAC extensions based on first-order logic.

We propose FORBAC, an extension of RBAC that incorporates the main features of different RBAC extensions from the literature, e.g. [5, 29, 34, 40]. FORBAC strikes a balance among the variety of policies it can express, the properties that can be verified, and its complexity, which is NP. Although a polynomial complexity would be desirable, we argue that NP-hardness cannot be avoided in policy analysis. To verify properties of FORBAC policies, we reduce them to satisfiability modulo theories and use the SMT-solver Z3 [21].

To evaluate our theses that (1) FORBAC is expressive enough for substantial real-world applications and (2) realistic policies can be analyzed with reasonable overhead, we conduct a case study on the access-control infrastructure of a major European bank. The bank's PDP manages around

350 applications, each with a separate security policy. In total, it manages access for close to 50,000 users and 57,000 actions. We give an overview of the bank's rules that govern both the assignments of roles to users and the assignment of permissions to roles. We express them as FORBAC policies and conduct experiments on a variety of relevant policy analysis queries. Using SMT solvers, most of the queries are answered in seconds. For a few of the queries, the evaluation takes several minutes and we identify reasons for this and suggest improvements.

The remainder of this chapter is organized as follows. In Section 2.2 we describe the features of different RBAC extensions from the literature and establish requirements for FORBAC. In Section 2.3 we define FORBAC's syntax and semantics and in Section 2.4 we show how to specify policy analysis queries for FORBAC policies. In Section 2.5 we present experimental results. In Section 2.6 we discuss related work and in Section 2.7 we draw conclusions.

2.2 Requirements for FORBAC

FORBAC is a RBAC extension designed to strike a balance between

- providing an expressive language for specifying RBAC policies and properties of these policies, and
- guaranteeing a low complexity for verifying policies against such properties.

Many extensions for RBAC like [34, 40, 52] include fragments of first-order logic and therefore make their policy analysis undecidable, or at least intractable. In the following, we present some of their central features to provide insight into the requirements for an expressive RBAC policy specification language. Based on these requirements, we define in Section 2.3 a fragment of first-order logic that is sufficiently simple, but still expressive enough to formalize realistic RBAC policies.

Attributes A powerful increase in RBAC's expressiveness is introduced by adding attributes to users, roles, and permissions. This allows fine-grained access control, e.g., providing a role for relationship mangers to access customer data but restricting each of their permissions to precisely those customers they manage. Instead of defining a separate role for each subset of customers, an attribute is added to the role relationship manager that specifies the set of customers assigned to it.

Extensions of roles with attributes have been proposed as *parameterized permissions* and *role templates* [3, 16, 29]. A parameterized permission describes a set of permissions that have certain attributes in common. For example, accessing customer data could be given as a parameterized permission *ReadData*(c), where c is a variable representing a customer. By using parameterized permissions, policy administrators only need to define one permission together with the *ReadData* attribute, instead of one permission for every customer. Role templates are sets of parameterized permissions. For example, for relationship manager we can define a role template *RelManager* containing the parameterized permission *ReadData*(c). When a user is assigned the role template *RelManager*, he or she is also assigned a set of customers C. The pair $(\mathit{RelManager}, C)$ is called a *role instance*. Assigned to this role instance, the user can access the data of every customer in C and therefore avoids creating a separate role for every relationship manager. We add parameterized permissions and role templates to our language and formalize them with functions and binary relations in first-order logic.

Role and permission assignments Moreover, RBAC extensions apply rules to assign roles to users and permissions to roles. Instead of manually administering these relations in large organizations where users and permissions change their attribute values frequently, machine-readable rules provide a convenient solution to create these relations automatically. Numerous RBAC extensions [29, 34, 38] use first-order logic to specify rules for user-role and role-permission assignment relations. However, they leave the fragment of first-order logic formalizing these specifications unrestricted.

We restrict the first-order fragment used for FORBAC. For instance, we exclude the arbitrary nesting of quantifiers. In practice, permissions simply require the presence or absence of values in the user's, role's, and permission's attributes.

Various logic-based policy specification language that are used in practice forbid quantifier alternation. For example, Lithium [31] does not allow quantifier alternation but it is still possible to express various parts of the U.S. legislation with it including fragments of the Privacy Rule. Another example is Cassandra [13], which also forbids quantifier alternation, but allows to formalize policies for the national electronic health record system of the United Kingdom.

Numeric constraints Constraints restricting dates and durations of permissions are often part of access control policies, such as the following fragment of the HIPAA rule shows:

> A period of creditable coverage shall not be counted, with respect to enrollment of an individual under a group health plan, if, after such period and before the enrollment date, there was a 63-day period during all of which the individual was not covered under any creditable coverage.

Moreover, functions in organizations are often limited to certain time periods or for a fixed duration. For example, vendors may access their contracts in the second week of every quarter of the year, and these contracts must be handed in within two weeks [14]. Hence, most of these constraints can be expressed by inequalities between integer values.

Note that there are other requirements that are used in access control specifications, but we excluded from our language. The most well-known of these features are role hierarchies [59], delegation [9], and separation-of-duty constraints [4], which we leave for future work.

Complexity of policy analysis To be usable in daily business, policy analysis of access control languages must be efficiently computable. However, in [19] it is shown that checking the basic query, if every access request is permitted, in a sufficiently expressive but simple policy specification language is NP-hard. State-of-the-art policy analysis tools like Margrave [54] and the one presented in [7] are used for analyzing realistic XACML policies like Continue [47] and are expressive enough to embed such a simple query in their policy analysis. As basic policy analysis queries are NP-hard, we believe P is too restrictive (unless P = NP) and therefore set the limit to perform our policy analysis to NP.

2.3 Syntax and semantics of FORBAC

Provided the requirements of our framework, we define the vocabulary of our policy specification language.

Definition 1. *A* FORBAC signature *is a triple* $\Sigma = (\mathcal{S}, \mathcal{A}_1, \mathcal{A}_2)$ *where* $\mathcal{S}$ *is a set of sorts* $\mathcal{S} = \mathcal{S}_{RBAC} \cup \{\textbf{\textit{Integer}}, \textbf{\textit{String}}\}$ *with,*

$$\mathcal{S}_{RBAC} = \{Users, Roles_1, Roles_2, \ldots, Roles_T, Perms\},$$

for $T \in \mathbb{N}$ *and* $\mathcal{A}_1$, $\mathcal{A}_2$ *are sets of unary function symbols. Every* $g \in \mathcal{A}_1 \cup \mathcal{A}_2$ *has a function type* $W_g \to V_g$ *with* $W_g \in \mathcal{S}_{RBAC}$*. For* $g \in \mathcal{A}_1$, $V_g \in \{\textbf{\textit{Integer}}, \textbf{\textit{String}}\}$ *and for* $g \in \mathcal{A}_2$, $V_g \in \{\mathcal{P}_f(\textbf{\textit{Integer}}), \mathcal{P}_f(\textbf{\textit{String}})\}$*, where* $\mathcal{P}_f(\textbf{\textit{Integer}})$ *and* $\mathcal{P}_f(\textbf{\textit{String}})$ *denote the sets of finite sets of integers and strings, respectively.*

Elements in $\mathcal{A}_1$ denote *single-valued attributes* and those in $\mathcal{A}_2$ denote *set-valued attributes*. We use the term *attribute* to refer to any single or set-valued attribute. We use $RT(\Sigma)$ to denote the set $\{Roles_1, \ldots, Roles_T\}$ of *role templates* of Σ.

Example 2. *We present a simple FORBAC-signature $\Sigma_B = (\mathcal{S}, \mathcal{A}_1, \mathcal{A}_2)$ for specifying the access control policies of an administrative tool to manage bank accounts. The sorts $\mathcal{S}$ are Users, $R_{\mathtt{Junior}}$, $R_{\mathtt{Senior}}$, Perms,* ***Integer****, and* ***String****. Here, $RT(\Sigma_B) = \{R_{\mathtt{Junior}}, R_{\mathtt{Senior}}\}$ where both represent two kinds of bank accounts, "junior accounts" and "senior accounts", respectively.*
The sort Users has the following four single-valued attributes

- *custID* : *Users* $\rightarrow$ ***String****,*
- *age* : *Users* $\rightarrow$ ***Integer****,*
- *home* : *Users* $\rightarrow$ ***String****,*
- *income* : *Users* $\rightarrow$ ***Integer****.*

where custID*,* age*, and* income *provide the corresponding real world attributes and* home *denotes the user's home country.*
The role template $R_{\mathtt{Senior}}$ has the single-valued attribute

- *limit* : $R_{\mathtt{Senior}} \rightarrow$ ***Integer***

that specifies the maximal amount a professional bank account holder may withdraw or transfer from his account.
The role template $R_{\mathtt{Junior}}$ has the set-valued attribute

- *region* : $R_{\mathtt{Junior}} \rightarrow \mathcal{P}_f$ (***String***)

that specifies in which countries a junior bank account holder can withdraw money without additional charges.
Perms have the following three single-valued attributes

- *service* : *Perms* $\rightarrow$ ***String****,*
- *value* : *Perms* $\rightarrow$ ***Integer****,*
- *country* : *Perms* $\rightarrow$ ***String****.*

The attribute service *describes the kind of transaction, e.g., withdrawing or transferring money from a bank account,* value *sets an upper bound to how much money may be involved, and* country *denotes the country where the transaction takes place.*

Definition 3. *Let $\Sigma = (\mathcal{S}, \mathcal{A}_1, \mathcal{A}_2)$ be a FORBAC signature. A* Σ-structure S *consists of*

- *a finite, non-empty set W^S, for each sort $W \in \mathcal{S}$, where **Integer**^{S} and **String**^{S} are the sets of integers and strings, and $\mathcal{P}_f(\textbf{Integer})^S$ and $\mathcal{P}_f(\textbf{String})^S$ are the sets of their finite subsets, respectively, and*
- *a function $g^S : W_g^S \to V_g^S$, for every $g \in \mathcal{A}_1 \cup \mathcal{A}_2$ with function type $W_g \to V_g$.*

We call the elements of Users^S users *in S, for a role template $R \in RT(\Sigma)$, we call the elements of R^S* role instances of R, *and we call the elements of Perms^S* permissions *of S.*

Example 4. *Let Σ_B be the FORBAC-signature from Example 2. Figure 2.2 shows a Σ_B-structure S with three users, two role instances of $R_{\texttt{Junior}}$, one role instance of $R_{\texttt{Senior}}$, and three permissions.*

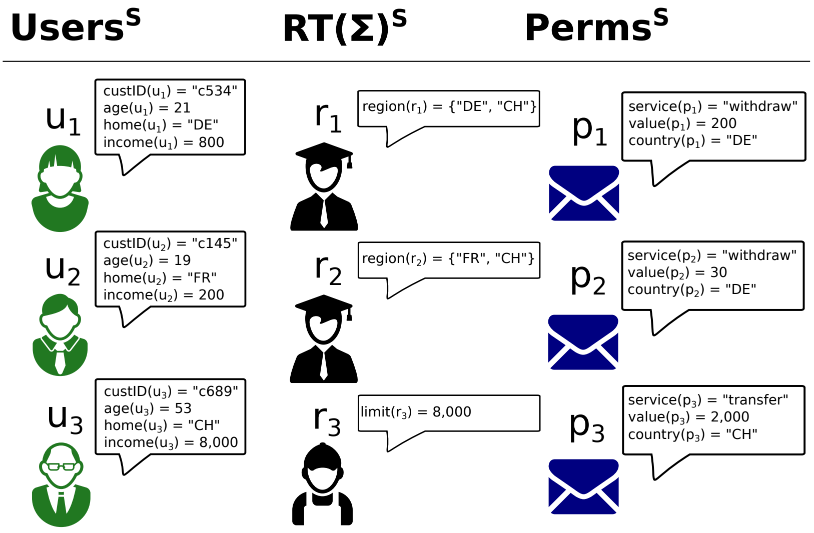


Figure 2.2: An example of a Σ_B-structure

Definition 5. *An* atomic FORBAC formula *can be either of the following expressions:*

- $t_1 \sim t_2$*, where* t_1 *and* t_2 *are* single-valued terms. *They are either constants of type **Integer** or **String**, or expressions of the form* $g(x)$*, where* g *is a single-valued attribute and* x *is a variable. The symbol* $\sim$ *can be* $=$, $\leq$, *or* $<$.
- $T_1 \propto T_2$*, where* T_1 *and* T_2 *are* set-valued terms. *They are either constant symbols denoting finite sets of strings, constant symbols denoting finite sets of finite intervals of integers, or expressions of the form* $G(x)$*, where* G *is a set-valued attribute and* x *is a variable. The symbol* $\propto$ *is either* $=$ *or* $\subseteq$.
- $t \in T$*, where* t *is a single-valued and* T *a set-valued term, respectively.*

The syntax of atomic FORBAC-formulas is summarized by the following BNF grammar:

$$\begin{array}{rl}\psi & ::= t \sim t \mid T \propto T \mid t \in T \\ t & ::= c \mid g(x) \\ T & ::= C \mid G(x) \\ \sim & ::= \leq \mid = \mid < \\ \propto & ::= \subseteq \mid = \end{array}$$

Here, c *ranges over integer and string constants,* g *ranges over single-valued attributes,* G *ranges over set-valued attributes,* C *is any finite set of strings or any finite set of integer intervals, and* x *is a variable of an appropriate type. Finally, a* FORBAC-formula *is a Boolean combination (negation, conjunction, disjunction, implication, or equivalence) of atomic FORBAC formulas.*

Definition 6. *The* size *of a FORBAC-formula* ϕ *is the number of occurrences of* ϕ*'s atomic FORBAC-formulas and is recursively defined by*

$$|\phi| = \begin{cases} 1 & \text{if } \phi \text{ is atomic} \\ |\psi| & \text{if } \phi \equiv \neg\psi \\ |\psi_1| + |\psi_2| & \text{if } \phi \equiv \psi_1 \bowtie \psi_2, \end{cases}$$

where $\bowtie \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

Remark 7. *Note that FORBAC can be embedded in many-sorted first-order logic. Given a FORBAC-formula, for every set-valued attribute* G*, choose a distinct binary relation symbol* R_G*. Rewrite every atomic FORBAC subformula containing a set-valued term as illustrated in the following three cases*

$$\begin{array}{rcl} t' \in G(t) & \rightsquigarrow & R_G(t,t') \\ G(t) \subseteq G'(t') & \rightsquigarrow & \forall y \,.\, (R_G(t,y) \rightarrow R_{G'}(t',y)) \\ G(t) = G'(t') & \rightsquigarrow & \forall y \,.\, (R_G(t,y) \leftrightarrow R_{G'}(t',y)) \end{array}$$

where G and G' range over set-valued attributes and t and t' range over single-valued terms. The remaining cases are analogous.

Definition 8. *A* FORBAC-policy *is a triple* $(\Sigma, \mathcal{UA}, \mathcal{PA})$*, where* Σ *is a FORBAC-signature. The* user-assignment specification

$$\mathcal{UA} = \bigcup_{R \in RT(\Sigma)} \mathcal{UA}_R(u,r)$$

and the permission-assignment specification

$$\mathcal{PA} = \bigcup_{R \in RT(\Sigma)} \mathcal{PA}_R(r,p)$$

are unions of sets $\mathcal{UA}_R(u,r)$ *and* $\mathcal{PA}_R(r,p)$ *of FORBAC-formulas over* Σ*, respectively. The* user-assignment formulas $\mathcal{UA}_R(u,r)$ *contain at most the two free variables u and r of sorts Users and R, respectively, and the* permission-assignment formulas $\mathcal{PA}_R(r,p)$ *contain at most the two free variables r and p of sorts R and Perms, respectively.*

Example 9. *Consider the FORBAC-signature* Σ_B *from Example 2 and suppose that we have the following policy:*

> *Users no older than 21 are assigned an instance of* $R_{\mathtt{Junior}}$*, which allows them to withdraw up to* 2,000 *CHF in the users' respective home country or in Switzerland.*
>
> *Users whose income exceeds* 7,500 *CHF are assigned an instance of* $R_{\mathtt{Senior}}$*, which allows them to withdraw and transfer money in any country provided the value does not exceed the user's income.*

The following FORBAC-policy $(\Sigma_B, \mathcal{UA}, \mathcal{PA})$ *formalizes this. AS* Σ_B *is given in Example 2, we only provide the formulas for* $\mathcal{UA}$ *and* $\mathcal{PA}$*.*

$$\mathcal{UA}_{R_{\mathtt{Junior}}}(u,r) \equiv \left(\begin{array}{l} age(u) \leq 21 \wedge \\ region(r) = \{home(u), \text{“}CH\text{”}\} \end{array} \right)$$

$$\mathcal{PA}_{R_{\mathtt{Junior}}}(r,p) \equiv \left(\begin{array}{l} service(p) \in \{\text{“}withdraw\text{”}\} \wedge \\ value(p) \leq 2{,}000 \wedge \\ country(p) \in region(r) \end{array} \right)$$

$$\mathcal{UA}_{R_{Senior}}(u,r) \equiv \left(\begin{array}{l} income(u) > 7{,}500 \wedge \\ limit(r) = income(u) \end{array} \right)$$

$$\mathcal{PA}_{R_{Senior}}(r,p) \equiv \left(\begin{array}{l} service(p) \in \{\text{“}withdraw\text{”}, \text{“}transfer\text{”}\} \wedge \\ value(p) \leq limit(r) \end{array} \right).$$

Let Σ be a FORBAC-signature, S a corresponding Σ-structure, and suppose $\overline{u}$ is a user, $\overline{p}$ a permission, and $\overline{r}$ a role instance of R in S. We say that $\overline{u}$ *is assigned* $\overline{r}$ if $\overline{u}$ and $\overline{r}$ satisfy $\mathcal{UA}_R(u,r)$ in S, and $\overline{r}$ *is assigned* $\overline{p}$ if $\overline{r}$ and $\overline{p}$ satisfy $\mathcal{PA}_R(r,p)$ in S, respectively.

Example 10. *Given the Σ_B-structure of Example 4, Figure 2.3 illustrates which permissions are assigned to which role instances and which role instances are assigned to which users.*

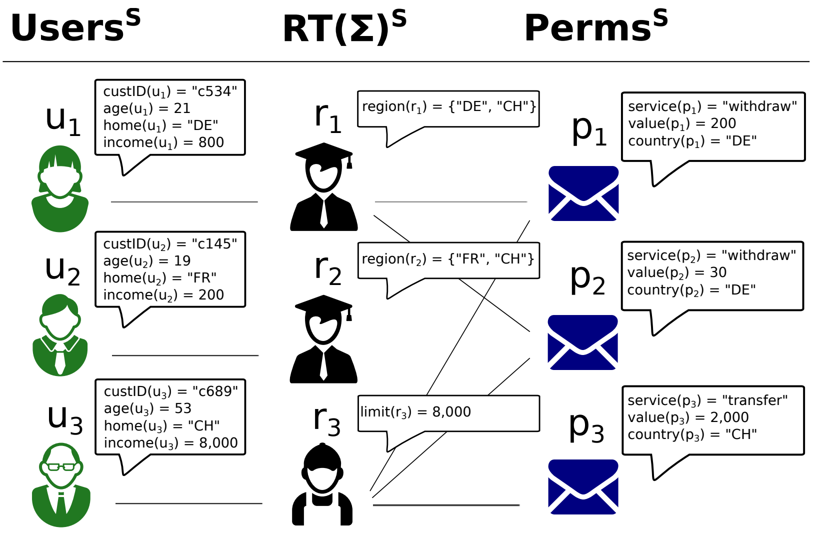


Figure 2.3: User and permission-assignments in the Σ_B-structure S

Definition 11. *Given a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ and a role template $R \in RT(\Sigma)$, the formula $Auth_R(u,p)$ is defined by*

$$\exists r \in R \,.\, \mathcal{UA}_R(u,r) \;\wedge\; \mathcal{PA}_R(r,p)$$

and $Auth(u,p)$ is defined by

$$\bigvee\nolimits_{R\in RT(\Sigma)} Auth_R(u,p)\,.$$

Given a Σ-structure S, we say that a user $\overline{u} \in Users^S$ is authorized for a permission $\overline{p} \in Perms^S$ *if $\overline{u}$ and $\overline{p}$ satisfy $Auth(u,p)$ in S.*

To simplify notation, we omit the sorts *Users* and *Perms* when quantifying over variables of their type, i.e., instead of writing

$$\forall u \in Users\, \exists r_1 \in R_1, r_2 \in R_2\,.\, \mathcal{UA}_{R_1}(u,r_1) \;\vee\; \mathcal{UA}_{R_2}(u,r_2),$$

we write

$$\forall u\, \exists r_1 \in R_1, r_2 \in R_2\,.\, \mathcal{UA}_{R_1}(u,r_1) \;\vee\; \mathcal{UA}_{R_2}(u,r_2).$$

Example 12. *Consider the FORBAC-signature presented in Example 2, the Σ_B-structure S presented in Example 4, and the FORBAC-policy presented in Example 9. User u_1 is authorized for permissions p_1 and p_2 and user u_3 is authorized for permissions p_1, p_2, and p_3.*

Instead of role templates one could use simply one sort role and a special type function to distinguish between different role templates. However, this makes specifying FORBAC-policies more complicated since attributes among role templates may differ. Policy administrators would need to use dummy values in case attributes are undefined for the respective role template. In Example 9 the attribute *region* is superfluous in the role template R_{Senior} and would need to be set to all existing countries.

It can be shown that authorization in a FORBAC-structure can be decided in polynomial time [19].

Theorem 13. *Given a FORBAC policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$, a Σ-structure S, a user $\overline{u} \in Users^S$, and a permission $\overline{p} \in Perms^S$, deciding whether $\overline{u}$ is authorized for $\overline{p}$ takes at most polynomial time.*

2.4 Policy analysis in FORBAC

We now define the language for posing analysis queries for FORBAC-policies. Since FORBAC-formulas can be expressed in first-order logic, this language is also a natural choice for reasoning about FORBAC-policies. However, first-order logic is undecidable in general and its restriction to fragments must be done with care. Halpern and Weissman [31] studied several fragments of first-order logic for specifying access control policies. They showed that even after limiting the number of quantifier alternations and removing function symbols, one can end up with a fragment where merely deciding authorization is intractable.

To strike a balance between expressiveness in property specification and efficiency in policy analysis, we propose the set of existential FORBAC-formulas as the language for specifying analysis queries.

Definition 14. *An* existential FORBAC-formula *is a first-order formula of the form* $\exists x_1, x_2 \ldots, x_n \,.\, \varphi(x_1, x_2, \ldots, x_n)$*, where* $\varphi(x_1, x_2, \ldots, x_n)$ *is a Boolean combination of FORBAC-formulas over a FORBAC-signature.*

To verify if a property holds for a FORBAC-policy, we build an existential FORBAC-formula that describes a countermodel that violates the property. The Boolean combination of FORBAC-formulas describes the negation of the property and the existential quantifiers specify the elements that should appear in a countermodel. The formula can then be input into an SMT solver, which attempts to find such a countermodel. The syntax of existential FORBAC-formulas limits quantifier alternation and the behavior of relations and functions so that deciding satisfiability is NP-complete. For a detailed proof we refer to [19].

Theorem 15. *Deciding the satisfiability of an existential FORBAC-formula is NP-complete.*

The low complexity, NP, is not for free. There are relevant policy analysis queries like observational equivalence and conflict [7] that cannot be expressed as existential FORBAC-formulas. However, they can be expressed in first-order logic and can be passed as input to an SMT-solver.

We present now four kinds of policy analysis queries and explain how to reduce them to satisfiability of existential FORBAC-formulas.

A *Authorization inspection* can be used to verify that a FORBAC-policy does not grant undesired access.

B *Assignment simplification* can be used to identify redundancies in FORBAC formulas.

C *Role subsumption* can be used to identify redundant role templates.

D *Redundant assignments* can be used to identify redundancies in the user-assignment relation.

These queries illustrate the expressive power of existential FORBAC formulas as a language for policy analysis for FORBAC. Moreover, they are all natural queries, that arise and require answers, when administrating policies specified in rich policy languages, e.g., where role templates and first-order user and permission-assignments interact.

2.4.1 Authorization inspection

Suppose we are given a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$, a FORBAC formula $\psi_{user}(u)$ with a free variable u of sort *Users*, and FORBAC formulas $\psi_1(p_1)$, $\psi_2(p_2), \ldots, \psi_k(p_k)$, with free variable $p_1, p_2, \ldots, p_k$ of sort *Perms*. Authorization inspection can be cast as the question of whether a formula of the following form is satisfiable:

$$\exists u, p_1, \ldots, p_k \,.\, \psi_{user}(u) \,\wedge\, \bigwedge_{i \le k} \big(\psi_i(p_i) \,\wedge\, Auth(u, p_i)\big)\,. \tag{2.1}$$

Checking this formula's satisfiability amounts to searching for a Σ-structure S with a user u who matches the criteria of ψ_{user} and who is authorized for some permissions $p_1, p_2, \ldots, p_k$ that match the criteria of $\psi_1, \psi_2, \ldots, \psi_k$, respectively.

Example 16. *Consider again the FORBAC-policy from Example 9. According to $\mathcal{UA}_{R_{\mathtt{Junior}}}(u, r)$, users can be assigned instances of $R_{\mathtt{Junior}}$ if they are at most 21 years old. Also, according to $\mathcal{PA}_{R_{\mathtt{Junior}}}(r, p)$, instances of $R_{\mathtt{Junior}}$ can never be granted permission to withdraw amounts larger than* 2,000 *CHF. One may conjecture that users who are at most 21 years old can never withdraw large amounts of money; they cannot, at least, for the Σ_B-structure in Figure 2.3. To determine whether this property holds for any Σ_B-structure, we instantiate Formula* (2.1) *as follows.*

$$\begin{aligned} \psi_{user}(u) &\equiv age(u) \le 21, \\ \psi_1(p_1) &\equiv \begin{pmatrix} service(p_1) = \text{“withdraw”} \wedge \\ value(p_1) > 2{,}000 \end{pmatrix}. \end{aligned}$$

The resulting instance of Formula (2.1) *is*

$$\exists u, p_1 \,.\, \begin{pmatrix} age(u) \le 21 \,\wedge \\ service(p_1) = \text{“withdraw”} \wedge \\ value(p_1) > 2{,}000 \,\wedge \\ Auth(u, p_1) \end{pmatrix}. \tag{2.2}$$

If this formula is unsatisfiable, then we have confirmed our conjecture. However, if we input this formula to the SMT-solver Z3, then Z3 outputs that it *is satisfiable and provides a model satisfying the formula. This model can be used to build a Σ_B-structure that refutes our conjecture.*

The following is a Σ_B-structure $\tilde{S}$ that satisfies Formula (2.2). Let Σ_B be the FORBAC-signature from Example 2. Figure 2.4 shows a Σ_B-structure $\tilde{S}$ with one user $\bar{u}$, one role instance $\bar{r}$ of type R_{Senior}, none of type R_{Junior}, and one permission $\bar{p}$. It is easy to see that $\bar{u}$ and $\bar{r}$ satisfy $\mathcal{UA}_{R_{Senior}}(u,r)$ and that

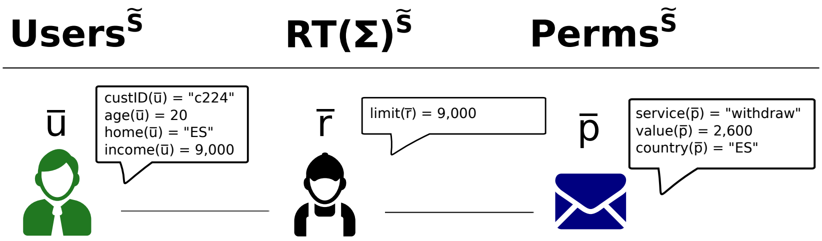


Figure 2.4: User and permission-assignments in the Σ_B-structure $\tilde{S}$

$\bar{r}$ and $\bar{p}$ satisfy $\mathcal{PA}_{R_{Senior}}(r,p)$. Therefore $\bar{u}$ is authorized for $\bar{p}$. This means that it is possible for users who are at most 21 years old to withdraw amounts greater than 2,000 *CHF. What they should do is to have an income greater than* 7,500 *CHF, so they obtain an instance $\bar{r}$ of R_{Senior} with a limit higher than* 7,500 *CHF. This would allow them to withdraw more than* 2,000 *CHF.*

Note that, as given, Formula (2.1) is not an existential FORBAC-formula because $Auth(u,p_i)$, for $i \leq k$, contains existential quantifiers. However, it can be rewritten into an existential FORBAC-formula by moving the existential quantifiers in $Auth(u,p_i)$, for $i \leq k$, to the front of the formula, using standard first-order equivalences.

2.4.2 Assignment simplification

Poor design or changes in policy specifications may lead to redundancies, which humans have difficulty detecting. We explain how we can identify redundancies using existential FORBAC formulas.

Example 17. *Consider the following FORBAC formula that specifies $\mathcal{UA}$ for some policy:*

$$\mathcal{UA}_R(u,r) \equiv \psi_1(u,r) \vee \psi_2(u,r),$$

where

$$\begin{aligned} \psi_1(u,r) &\equiv unit(r) = 45 \wedge level(u) = 23 \text{ and} \\ \psi_2(u,r) &\equiv unit(r) = 45 \wedge level(u) > 20. \end{aligned}$$

$\mathcal{UA}_R(u,r)$ consists of a disjunction of two formulas, where the satisfaction of the first formula implies the satisfaction of the second one. This means that $\psi_1(u,r)$ is redundant. To confirm this, we can show that the following formula is valid:

$$\forall u \forall r \in R\,.\,(\psi_1(u,r) \;\vee\; \psi_2(u,r)) \longleftrightarrow \psi_2(u,r)\,.$$

This is equivalent to showing that the following existential FORBAC formula

$$\exists u, r \in R\,.\,\neg((\psi_1(u,r) \;\vee\; \psi_2(u,r)) \longleftrightarrow \psi_2(u,r))$$

is not satisfiable.

The same technique can be used to detect redundancies in $\mathcal{PA}_R(r,p)$. In general, whenever one conjectures that a FORBAC formula $\psi(x_1, x_2, \ldots, x_k)$ is equivalent to another formula $\psi'(x_1, x_2, \ldots, x_k)$, one can check this by determining whether the following formula is valid:

$$\forall x_1, x_2, \ldots, x_k\,.\,\psi(x_1, x_2, \ldots, x_k) \longleftrightarrow \psi'(x_1, x_2, \ldots, x_k)\,.$$

This is equivalent to determining whether the following existential FORBAC formula is unsatisfiable:

$$\exists x_1, x_2, \ldots, x_k\,.\,\neg\big(\psi(x_1, x_2, \ldots, x_k) \longleftrightarrow \psi'(x_1, x_2, \ldots, x_k)\big)\,.$$

2.4.3 Role subsumption

RBAC systems used in large enterprises with multiple administrators may end up with equivalent redundant roles, especially, when the administrators are unaware of roles previously created by other administrators. Identifying these roles helps simplify RBAC policies. We explain how this situation can occur in FORBAC.

Example 18. *Consider a FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ with $RT(\Sigma) = \{R_1, R_2\}$ and role permission-assignment formulas*

$$\mathcal{PA}_{R_1}(r,p) \;\equiv \left(\begin{array}{l} action(p) \in \{\text{“}read\text{”}, \text{“}write\text{”}\} \;\wedge \\ (level(r) = level(p) \;\vee\; level(r) > level(p)) \end{array}\right)$$

$$\mathcal{PA}_{R_2}(r,p) \;\equiv \left(\begin{array}{l} (action(p) = \text{“}read\text{”} \;\vee\; action(p) = \text{“}write\text{”}) \;\wedge \\ level(r) \geq level(p) \end{array}\right).$$

Now, consider a Σ-structure S with two role instances $\overline{r_1}$ and $\overline{r_2}$ of R_1 and R_2, respectively. Suppose that $level^S(\overline{r_1}) = level^S(\overline{r_2})$. Observe that both instances are assigned the same set of permissions. As a result, whenever a user is assigned an instance $\overline{r}$ of R_2, she can be assigned instead an instance $\overline{r}'$ of R_1 with $level^S(\overline{r}') = level^S(\overline{r})$. The user would be authorized for the same set of permissions. Hence R_2 is redundant.

We now formally define the ideas from the previous example. For simplicity, we ignore set-valued attributes, but the presentation is analogous for the general case.

Definition 19. *Let Σ be a FORBAC-signature and let $R_1, R_2 \in RT(\Sigma)$. We say that R_1* expands *R_2 if for every attribute g of type $R_2 \to W$, with $W \in \{$**String**, **Integer**$\}$, there is the same symbol g, but of type $R_1 \to W$.*

Definition 20. *Let R_1 and R_2 be two role templates in a FORBAC-signature. We say that R_1* subsumes *R_2 and R_1* expands *R_2 if the following formula holds:*

$$\forall r_1 \in R_1, r_2 \in R_2 \,.\, \bigwedge\nolimits_{g:R_2 \to W} g(r_1) = g(r_2) \to \\ \forall p \,.\, \mathcal{PA}_{R_2}(r_2, p) \to \mathcal{PA}_{R_1}(r_1, p)\,.$$

Here, g ranges over attributes of type $R_2 \to W$, with $W \in \{\textbf{String}, \textbf{Integer}\}$. This formula says the following. Let $\overline{r}_1$ and $\overline{r}_2$ be two role instances of R_1 and R_2, respectively. If $g^S(\overline{r}_1) = g^S(\overline{r}_2)$, for every attribute g of type $R_2 \to W$, then any permission assigned to $\overline{r}_2$ is also assigned to $\overline{r}_1$.

Using first-order logic equivalencies, it is easy to prove that R_1 subsumes R_2 iff R_1 expands R_2 and the following existential FORBAC-formula is unsatisfiable:

$$\exists\, r_1 \in R_1, r_2 \in R_2\, \exists p \,.\, \bigwedge\nolimits_{g:R_2 \to W} g(r_1) = g(r_2) \land \\ \mathcal{PA}_{R_2}(r_2, p) \land \neg\mathcal{PA}_{R_1}(r_1, p)\,.$$

Finally, we call two roles *equivalent* if they subsume each other. Equivalent roles point to potential redundancies in the policy. However, we note that two equivalent roles are not necessarily redundant. It may happen that such roles have different functions from an organizational perspective. For example, the role of a programmer may have exactly the same types of permissions as the role of a tester, but they need to be distinguished in an organization [6].

2.4.4 Redundant assignments

In classical RBAC, the assignment of roles to users and the assignment of permissions to roles are two tasks performed by different people who do not necessarily communicate with each other. The assignment of roles may be performed, for example, by people in human resources; whereas the assignment of permissions may be performed by the application owners. This might lead to a situation where for two roles r_1 and r_2 the permissions assigned to r_2 are contained in those assigned to r_1 and the users who are assigned r_2 are also assigned r_1. This is illustrated in Figure 2.1 in the introduction. In this case, role r_2 might be redundant.

This situation, presented in [6], occurs in a kind of FORBAC policies that we call *functional FORBAC-policies*. In a functional FORBAC-policy, for every role template R, any two role instances assigned to a same user have exactly the same attribute values. The policy presented in Example 9 is a functional FORBAC policy. For any two role instances $\overline{r}$ and $\overline{r}'$ of $R_{\texttt{Junior}}$ assigned to a user $\overline{u}$, we have that $region(\overline{r}) = region(\overline{r}') = \{home(\overline{u}), \text{“CH”}\}$. Similarly, for any two instances $\overline{r}$ and $\overline{r}'$ of $R_{\texttt{Senior}}$ assigned to a user $\overline{u}$, we have that $limit(\overline{r}) = limit(\overline{r}') = income(\overline{u})$. Contrast this with a FORBAC-policy with a role template R such that

$$\mathcal{UA}_R(u,r) \;\equiv\; age(u) \geq 18 \;\wedge\; level(r) \leq 50.$$

This is not a functional FORBAC-policy. A user over 18 can be assigned several role instances, each with a different value for *level*. We now formally define functional FORBAC-policies.

Definition 21. *A FORBAC-policy* $(\Sigma,\mathcal{UA},\mathcal{PA})$ *is* functional *if every role template* $R \in RT(\Sigma)$ *satisfies the following two requirements:*

1. $\mathcal{UA}_R(u,r)$ *can be written as a conjunction* $\mathcal{UA}^u_R(u) \wedge \mathcal{UA}^r_R(u,r)$. *This means that the conditions for assigning a role instance to a user can be split in two: requirements the user must fulfill and requirements that the role instance must fulfill based on the user attributes.*

2. *The following formula is valid:*

$$\forall u \forall r \in R, r' \in R.\, \mathcal{UA}^r_R(u,r) \;\wedge\; \mathcal{UA}^r_R(u,r') \rightarrow \\ \bigwedge\nolimits_{g:R\rightarrow W} g(r) = g(r'),$$

*where g ranges over attributes of type $R \rightarrow W$, with $W \in$ {**Integer**, **String**, $\mathcal{P}_f$(**Integer**), $\mathcal{P}_f$(**String**)}. This means that any two role instances of R assigned to a same user have the same attribute values.*

It is easy to automatically check if a FORBAC-policy $(\Sigma,\mathcal{UA},\mathcal{PA})$ is functional. To check the second requirement, one checks, for every $R \in RT(\Sigma)$, whether the following existential FORBAC-formula is unsatisfiable.

$$\begin{array}{l} \exists u \, \exists r \in R, r' \in R \,. \\ \quad \mathcal{UA}^r_R(u,r) \ \wedge \ \mathcal{UA}^r_R(u,r') \ \wedge \ \bigvee_{g:R\rightarrow W} g(r) \neq g(r') \,. \end{array}$$

Having defined what a functional FORBAC-policy is, we now introduce a policy analysis query for identifying redundant formulas in $\mathcal{UA}$. We start with an example.

Example 22. *Consider a FORBAC-policy* $(\Sigma, \mathcal{UA}, \mathcal{PA})$ *with* $RT(\Sigma) = \{R_1, R_2\}$ *and* $\mathcal{UA}$ *and* $\mathcal{PA}$ *given by*

$$\mathcal{UA}_{R_1}(u,r) \ \equiv \ age(u) \geq 18 \ \wedge \ level(r) = 6$$

$$\mathcal{UA}_{R_2}(u,r) \ \equiv \ age(u) \geq 21 \ \wedge \ level(r) = 5$$

$$\begin{array}{l} \mathcal{PA}_{R_1}(r,p) \ \equiv \\ \quad \left(\begin{array}{l} action(p) \in \{\text{“read”}, \text{“write”}\} \ \wedge \\ level(r) \geq level(p) \end{array} \right) \end{array}$$

$$\begin{array}{l} \mathcal{PA}_{R_2}(r,p) \ \equiv \\ \quad \left(\begin{array}{l} action(p) \in \{\text{“read”}\} \ \wedge \\ level(r) \geq level(p) \end{array} \right) . \end{array}$$

Note that this is a functional FORBAC-policy. Now, observe that whenever a user is assigned an instance $\overline{r_2}$ *of* R_2*, he is also assigned an instance* $\overline{r_1}$ *of* R_1*. Moreover,* $\overline{r_1}$ *would get more permissions than* $\overline{r_2}$*. This implies that* $\mathcal{UA}_{R_2}(u,r)$ *is redundant.*

We now formally define this policy analysis query. Given a functional FORBAC-policy $(\Sigma, \mathcal{UA}, \mathcal{PA})$ and two role templates $R_1, R_2 \in RT(\Sigma)$, we want to check if the following formula is valid:

$$\begin{array}{l} \left(\forall u \,. \mathcal{UA}^u_{R_2}(u) \rightarrow \mathcal{UA}^u_{R_1}(u)\right) \wedge \\ \quad \forall u \forall r_1 \in R_1, r_2 \in R_2 . \\ \qquad \mathcal{UA}^r_{R_1}(u, r_1) \ \wedge \ \mathcal{UA}^r_{R_2}(u, r_2) \rightarrow \\ \qquad\quad \forall p \,. \mathcal{PA}_{R_2}(r_2, p) \rightarrow \mathcal{PA}_{R_1}(r_1, p) \,. \end{array}$$

If the previous formula is valid, then $\mathcal{UA}_{R_2}(u,r)$ is redundant in the FORBAC-policy. Checking the validity of the previous formula is equivalent to checking whether the following existential FORBAC-formula is unsatisfiable:

$$\begin{aligned}
&\big(\exists u\,.\,\mathcal{UA}^u_{R_2}(u) \;\wedge\; \neg\mathcal{UA}^u_{R_1}(u)\big) \vee \\
&\qquad \exists u \exists r_1 \in R_1, r_2 \in R_2 \exists p\,. \\
&\qquad\quad \mathcal{UA}^r_{R_1}(u, r_1) \;\wedge\; \mathcal{UA}^r_{R_2}(u, r_2) \;\wedge \\
&\qquad\quad \mathcal{PA}_{R_2}(r_2, p) \;\wedge\; \neg\mathcal{PA}_{R_1}(r_1, p)\,.
\end{aligned}$$

2.5 Experimental results

We present here the evaluations of our two theses: the FORBAC language is suitable for specifying realistic access control policies and these policies can be analyzed with reasonable overhead. For this, we conducted a case study on the access-control infrastructure of a European bank. We had access to the access control policies of 350 applications, in particular the rules defining the assignments of roles to users and the assignments of permissions to roles. We chose 10 of those policies and translated their rules into FORBAC-policies. In our translation, we omitted those parts dealing with delegation of role instances and separation-of-duty constraints, which are out of the scope of FORBAC, as explained in Section 2.2. For each of the 10 resulting FORBAC-policies, we randomly generated 10 different instances of the problems from Section 2.4 and checked them against their respective policies using Z3.

2.5.1 Policy structure

We now describe the structure of the 10 translated FORBAC-policies.

User-assignment relation In the FORBAC-policies, users are assigned role instances in two different ways. First, depending on a user's attribute values, like *job* or *country*, the user is automatically assigned the role instances that allow him to perform his duties. Second, users may require for some tasks more role instances than what the bank's policy automatically assigns to them. They therefore request additional role instances from the policy administrator, who assigns them individually.

These two ways are called *provisioned* and *individual* assignments. Both can be expressed as FORBAC-formulas of the following form:

$$\left(\bigwedge_i \mathit{conditions}_i(u)\right) \wedge \mathit{instanceAssigned}(u, r). \tag{2.3}$$

Whenever a user's attribute values satisfy $\bigwedge_i \mathit{conditions}_i(u)$, then the user can be assigned a role instance whose attribute values are defined by *instanceAssigned*(u, r). An example of this formula is

$$(\mathit{job}(u) = \text{“trader”} \;\wedge\; \mathit{country}(u) \in \{\text{“FR”}, \text{“USA”}\}) \\ \wedge\; \mathit{location}(r) = \mathit{country}(u) \;\wedge\; \mathit{value}(r) = 10{,}000.$$

This expresses a *provisioned* assignment, which assigns users, who are traders working in France or the USA, to a role instance with their own country as location and a value of $10,000$.

Individual assignments are a special case of *provisioned* assignments, where $\bigwedge_i \mathit{conditions}_i(u)$ contains only one conjunct of the form *userID*$(u) = c$, with c a constant, and *instanceAssigned*(u, r) does not contain any attribute of the sort *User*. An example of an *individual* assignment is

$$\mathit{userID}(u) = 73{,}134 \;\wedge\; \mathit{location}(u) = \text{“FR”} \;\wedge\; \text{value}(r) = 10{,}000.$$

Here, *userID* is an attribute of type *Users* $\rightarrow$ **Integer** used to identify the application's users.

The provisioned and the individual assignments for each role template $R \in RT(\Sigma)$ are expressed in $\mathcal{UA}_R(u, r)$ as a large disjunction of FORBAC-formulas of the form (2.3). To differentiate between provisioned and individual assignments we partition the disjunctions

$$\mathcal{UA}_R(u, r) \;\equiv\; \mathcal{UA}_R^1(u, r) \;\vee\; \mathcal{UA}_R^2(u, r)$$

in two parts, where $\mathcal{UA}_R^1(u, r)$ contains the provisioned assignments and $\mathcal{UA}_R^2(u, r)$ contains the individual assignments.

Permission-assignment relation For a role template R, the formula $\mathcal{PA}_R(r, p)$ has the form

$$\bigvee_i \bigwedge_j f_{ij}(p) \sim g_{ij}(r),$$

where f_{ij} and g_{ij} are attributes and $\sim$ is one of the following: $=, \neq, \in,$ or $\notin$.

Size of the policies In Table 2.1, we report on the size of the 10 translated FORBAC-policies. The label $\sharp\mathcal{A}$ denotes the number of (single- and set-valued) attributes in the signature . $|\mathcal{UA}^1|$ denotes $\sum_R \left|\mathcal{UA}_R^1(u, r)\right|$, $\left|\mathcal{UA}^2\right|$ denotes $\sum_R \left|\mathcal{UA}_R^2(u, r)\right|$ and $|\mathcal{PA}|$ denotes $\sum_R |\mathcal{PA}_R(u, r)|$. $\sharp$*Users* is the number of users. Finally, $\sharp RT(\Sigma)$ is the number of role templates.

2.5.2 Generating instances of queries

For each of the selected policies and for each query presented in Section 2.4, we generated 10 instances and verified them against the selected policy with Z3. We present next, for each query, how we generated these instances.

Policy	$\sharp\mathcal{A}$	$\lvert\mathcal{UA}^1\rvert$	$\lvert\mathcal{UA}^2\rvert$	$\lvert\mathcal{PA}\rvert$	$\sharp Users$	$\sharp RT(\Sigma)$
App1	19	33	238,052	126	3,490	6
App2	24	1,646	174,655	1668	9,330	96
App3	56	694	256,439	232	34,782	51
App4	20	78	135,089	262	17,554	11
App5	20	16	3,262	156	85,949	8
App6	9	56	4,451	200	23,368	17
App7	15	363	1,911	237	44,276	14
App8	36	318	13,144	661	20,438	14
App9	15	249	9,427	160	8,152	11
App10	34	46	1,734	120	24,199	12

Table 2.1: Size of the FORBAC policies for 10 bank applications

Authorization inspection We generate each instance as follows.

1. Build $\psi_{user}(u)$. Randomly choose an attribute g of type $Users \rightarrow W$, with $W \in \{\textbf{Integer}, \textbf{String}\}$ and a constant value c of type W. Let $\psi_{user}(u)$ be $g(u) = c$.

2. Randomly choose a value $k \leq 10$.

3. for $i = 1$ to k,

   (a) Build $\psi_i(p_i)$. Randomly choose up to 5 attributes g_1, $g_2, \ldots,$ g_5 and constant values c_1, $c_2, \ldots,$ c_5. Let

   $$\psi_i(p_i) \equiv \bigwedge_{i \leq 5} g_i(p_i) = c_i.$$

   (b) Build $Auth(u, p_i)$ as

   $$\bigvee_R \exists r \in R \,.\, \mathcal{UA}_R(u, r) \wedge \mathcal{PA}_R(r, p_i),$$

4. The instance is the formula

$$\exists u \, \exists p_1, \ldots, p_k \,.\, \psi_{user(u)} \wedge \left(\bigwedge_{i \leq k} \psi_i(p_i) \wedge Auth(u, p_i) \right).$$

Assignment simplification Recall that Σ is the FORBAC-signature used to specify the application's FORBAC-policy. Recall too that, for a role template R in Σ, $\mathcal{UA}_R^1(u,r)$ is a disjunction of formulas of the form

$$\left(\bigwedge_{i\leq K} \mathit{conditions}_i(u)\right) \wedge \mathit{instanceAssigned}(u,r).$$

To generate an instance of the assignment simplification query, we randomly choose $j \leq K$. The generated instance is then

$$\exists u\,.\,\neg\left(\bigwedge_{i\leq K} \mathit{conditions}_i(u) \longleftrightarrow \bigwedge_{\substack{i\leq K\\ i\neq j}} \mathit{conditions}_i(u)\right).$$

Intuitively, we want to know if $\mathit{conditions}_j(u)$ is redundant. Note that $\mathit{instanceAssigned}(u,r)$ is not part of the formula because we want to simplify the conditions that assign the role to a user and $\mathit{instanceAssigned}(u,r)$ just describes the role instance.

Role subsumption To generate an instance of the role subsumption query, we randomly choose two role templates R_1 and R_2. We then take $\mathcal{PA}_{R_1}(r,p)$ and $\mathcal{PA}_{R_2}(r,p)$ and define the instance as

$$\begin{aligned}&\exists r_1 \in R_1, r_2 \in R_2\ \exists p\,.\\ &\quad \textstyle\bigwedge_g g(r_1)=g(r_2)\ \wedge\ \mathcal{PA}_{R_2}(r_2,p)\ \wedge\ \neg\mathcal{PA}_{R_1}(r_1,p)\,.\end{aligned}$$

Intuitively, we ask if R_1 subsumes R_2. The bank's policies, once translated in FORBAC, have the following property: for any two role templates R_1 and R_2, if there is defined a function $g : R_1 \to W$, then there is also defined a function $g : R_2 \to W$. In other words, every attribute is defined for all role templates, so it follows immediately that any two role templates expand each other.

Redundant assignments Recall that, for every role template R, $\mathcal{UA}_R^1(u,r)$ is a disjunction of formulas of the form

$$\left(\bigwedge_{i} \mathit{conditions}_i(u)\right) \wedge \mathit{instanceAssigned}(u,r).$$

These disjuncts are always functional FORBAC-formulas. To generate an instance of the redundant assignments query, we randomly choose two role templates R_1 and R_2 and one disjunct from each of $\mathcal{UA}^1_{R_1}(u,r)$ and $\mathcal{UA}^1_{R_2}(u,r)$. Let

$$\left(\bigwedge_i conditions_i(u)\right) \wedge instanceAssigned(u,r), \text{ and}$$
$$\left(\bigwedge_j conditions'_j(u)\right) \wedge instanceAssigned'(u,r)$$

be the selected disjuncts. The instance is

$$\begin{aligned}&\left(\exists u \,.\, \textstyle\bigwedge_i conditions_i(u) \wedge \neg \bigwedge_j conditions'_j(u)\right) \vee \\ &\exists u\ \exists r_1 \in R_1, r_2 \in R_2\ \exists p\,. \\ &\quad instanceAssigned(u,r_1) \wedge \\ &\quad instanceAssigned'(u,r_2) \wedge \\ &\quad \mathcal{PA}_{R_2}(r_2,p)\ \wedge\ \neg\mathcal{PA}_{R_1}(r_1,p)\,.\end{aligned}$$

2.5.3 Results and conclusions

We used the SMT solver Z3 to verify each of these instances against the corresponding FORBAC-policy. We ran the checks on a 2.50 GHz Intel Core i5 CPU, with 8GB of RAM. Table 2.2 shows how much time Z3 took to verify, for each application, 10 instances of the queries of Authorization Inspection (AI), Assignment Simplification (AS), Role Subsumption (RS), and Redundant Assignments (RA). A cell with NA indicates that there were not enough provisioned assignments to create 10 instances for the corresponding policy. A cell with > 360 indicates that Z3 required more than 60 minutes for 10 instances, i.e. more than 360 seconds on average. Regarding our first thesis, expressiveness, we could express the main parts

Policy	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
AI	357.87	>360	>360	>360	2.98	1.85	3.52	32.69	38.02	0.30
AS	0.61	0.63	0.57	0.54	*NA*	0.75	0.87	0.5	0.49	*NA*
RS	0.53	0.55	0.43	0.43	0.45	0.47	0.46	0.47	0.47	0.44
RA	0.73	0.47	0.46	0.49	*NA*	0.58	0.53	0.59	0.49	*NA*

Table 2.2: Time (in seconds) needed by Z3 for an instance on average

of the policies in FORBAC, except for delegation of role instances and separation of duty constraints. Regarding our second thesis, that policies can be analyzed with reasonable overhead, the instances we generated for AS, RS, and RA can be analyzed by Z3 within one second for realistic policies. The only exceptions are the first four applications in AI, which include many individual assignments.

For policies where many individual assignments must be analyzed, we see two ways of proceeding in practice:

1. Check the AI queries only on the provisioned assignments $\mathcal{UA}_R^1(u,r)$. We note that in practice $\mathcal{UA}_R^2(u,r)$ should not be too large and can often be replaced by provisioned assignments to improve policy maintenance.

2. In case $\mathcal{UA}_R^2(u,r)$ is large, evaluate AI on each individual assignment separately. Based on our experience it is possible to restrict the query to a proper subset of all individual assignments before creating the SMT-files.

If we evaluate the 10 instances on $\mathcal{UA}_R^1(u,r)$ only, as described in 1), we obtain the following average times:

Policy	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
AI^1	0.11	4.97	1.56	0.15	0.13	0.12	0.40	0.45	0.31	0.12

Alternatively if we evaluate as in 2) on 10 randomly chosen users with individual assignments, we obtain the following average times:

Policy	App1	App2	App3	App4	App5	App6	App7	App8	App9	App10
AI^2	0.97	6.73	1.75	0.96	0.89	0.78	0.87	1.21	0.95	2.02

Finally note that the evaluation of the individual assignments can be executed in parallel on a cluster since they can be analyzed independently. If parallelization is not supported, one can instead proceed iteratively. In each of the 10 FORBAC-policies, there were at most 10,000 users with individual assignments. Therefore, an upper bound on the time required for 2) is one day in the worst case. From the bank's point of view this is still reasonable since AI queries must be executed only rarely for audits and can be run offline over night.

2.6 Related work

In the early days of RBAC, it was sufficient to propose an RBAC model that overcame the limitations that were observed in previous RBAC models. Here, limitations were understood in terms of expressiveness, not policy analysis. Later, some authors (such as [38] and [28]) noted that the expressive power of policy specification languages hindered policy analysis. Hence, our perspective is that new RBAC models should be frameworks that provide three components: a language for policy specification, a language for specifying policy analysis queries, and procedures for efficiently evaluating these queries. To the best of our knowledge, there are two such frameworks that have gained acceptance in the literature: [55] and [7]. We discuss them as well as other work related to RBAC policy analysis.

2.6.1 Margrave

Margrave [55] is a framework for policy specification and analysis. With Margrave, users can specify policies and query their properties. Margrave then searches for representative scenarios that satisfy the property.

In our framework, we reason about RBAC policies and compute one satisfying scenario rather than a set of scenarios. In contrast to Margrave, our framework can reason about integer constraints and, therefore, express policies like “Alice can read a file if her clearance level is greater than the file’s clearance level”.

FORBAC’s interaction between set-valued terms and roles gives rise to policies that cannot be modeled in Margrave. Consider, for example, a FORBAC signature with a role template *Technician* and the following functions:

- *OU* that assigns a string to each user which represents the user’s organizational unit.
- *Sectors* that assigns a set of integers to each role instance of *Technician*.
- *sector* that assigns an integer to each permission.

Suppose that only those users u with $\mathit{OU}(u) = A$ are assigned an instance r of *Technician* with $\mathit{Sectors}(r) = [100, 110]$, and that an instance r of *Technician* is authorized to any permission p with $\mathit{sector}(p) \in \mathit{Sectors}(r)$.

In Margrave, we can use two policies to model the user and permission-assignments, respectively. The first policy indicates when a user u is assigned a role instance r and can be expressed in Margrave as the following first-order formula:

$$\Phi_{UA} \equiv \forall u, r \,.\, \left(\begin{array}{l} OU(u) = A \wedge \\ \bigwedge_{100 \leq i \leq 110} Sectors(r, i) \end{array} \right) \rightarrow mUA(u, r).$$

The second policy indicates when a role instance r is authorized for permission p:

$$\Phi_{PA} \equiv \forall r, p \,.\, Sectors(r, sector(p)) \rightarrow mPA(r, p).$$

To decide authorization, we use the Margrave policy analysis framework. A user u is authorized for permission p if the following query is satisfiable:

$$\exists r \,.\, \Phi_{UA} \wedge \Phi_{PA} \wedge mUA(u, r) \wedge mPA(r, p).$$

Now, consider the following property: There is a user who is authorized for a permission in sector 300. Such a user does not exist since instances of role *Technician* can access only the sectors between 100 and 110. However, if we query Margrave with

$$\exists u, r, p \,.\, mUA(u, r) \wedge mPA(r, p) \wedge sector(p) = 300,$$

then Margrave responds with a scenario consisting of a user $\tilde{u}$ with $OU(\tilde{u}) = A$, a permission $\tilde{p}$ with $sector(\tilde{p}) = 300$, and a role instance $\tilde{r}$ with $Sectors(\tilde{r}) = \{100, \ldots, 110, 300\}$. This is because the policy that assigns instances of *Technician* allows one to assign to a user any instance that contains at least the sectors from 100 through 110. It is not possible to assign an instance of *Technician* that contains only the sectors from 100 through 110 unless we explicitly say that all other sectors must not be assigned:

$$\left(\begin{array}{l} OU(u) = A \wedge \\ \bigwedge_{100 \leq i \leq 110} Sectors(r, i) \wedge \\ \bigwedge_{i \geq 111} \neg Sectors(r, i) \end{array} \right) \rightarrow mUA(u, r). \tag{2.4}$$

Such specifications, however, are difficult to maintain. If new sectors are created in the environment, then the policy must be adapted to prevent the new sectors from being authorized for technicians. In addition, Margrave cannot efficiently handle policies with large attribute domains [7]. In contrast, this policy can be expressed in FORBAC as follows:

$$\begin{aligned} \mathcal{UA}_{Technician}(u, r) &\equiv \left(\begin{array}{l} OU(u) = A \wedge \\ Sectors(r) = [100, 110] \end{array} \right) \\ \mathcal{PA}_{Technician}(r, p) &\equiv sector(p) \in Sectors(r). \end{aligned} \tag{2.5}$$

The fact that no user is authorized for permission in sector 300 follows from the unsatisfiability of the existential FORBAC formula:

$$\exists u, p\,.\, Auth(u,p)\ \wedge\ sector(p) = 300.$$

2.6.2 Athena+Yices

Another framework related to our work is Athena+Yices [7], which merges functional programming with first-order logic for both policy specification and property verification. For verifying properties, the Yices SMT-solver is used. They do experiments on the CONTINUE [47] policy and achieve results better than Margrave [28] and the framework used in [46]. In contrast to FORBAC, their language can express arithmetic constraints and they can reason about XACML policies.

Athena+Yices faces the same problem as Margrave when dealing with the administration of set-valued attributes. Although Athena+Yices can reason about arithmetic constraints, they do not allow quantification when specifying policies. This makes it impossible to write Policy (2.5), unless users explicitly list all the sectors that should and should not be allowed.

The authors [7] do not provide complexity bounds for the policy analysis problems they consider. The policy properties they propose are undecidable in general because their syntax allows addition, subtraction, and multiplication of integer variables, which allows Diophantine equations to be expressed as requirements for authorization. This shows again how one ends up in undecidable fragments of policy analysis if the language is not restricted. In contrast, FORBAC ensures that all the given policy analysis queries are evaluated in NP. However, the low complexity of policy analysis for FORBAC does not come for free. Existential FORBAC formulas cannot express relevant queries presented in [7] like observational equivalence, conflict detection, and change-impact analysis.

2.6.3 Other related work

Other researchers examine policy analysis for RBAC, but they neither propose a language for specifying properties nor procedures for verifying them. They only propose procedures for verifying specific properties.

For example, [27, 37, 62] focus on the reachability problem. In this problem, a set of users is given together with a set of administrative rules. These administrative rules specify who can assign and remove role assignments according to the roles assigned to the users. The objective of the reachability problem is to decide if it is possible for the given set of users to reach a goal set of roles using the administrative rules. We did not include the reachability problem in our framework because there are many efficient frameworks proposed for this problem. Additionally, the reachability problem is defined only for classical roles. To the best of our knowledge, no work has investigated administrative rules for role templates.

[8] uses SMT-solvers to identify induced role hierarchies. They model rule-based user-role assignments and decide which role assignments are entailed by others within an RBAC model. Their framework can reason about negative authorization via negated roles. However, they only work with classical roles and single-valued attributes. In contrast, we analyze this problem with role templates and set-valued attributes.

Another language that combines policy specification and policy analysis is presented in [24]. It is a Datalog-based language that can express policies in dynamic environments and performs verification on goal reachability and contextual policy containment. Goal reachability consists of deciding if the system can reach a state where a given property holds, expressed as an existential formula. Contextual policy containment consists of deciding, for two given policies and in every state the system can reach, whether those permissions authorized by one policy are authorized by the other. However, their framework neither expresses nor reasons about numeric constraints and set-valued attributes.

Lithium [31] is a policy specification language based on a fragment of first-order logic. Lithium policies are sets of formulas consisting of either conjunctions of ground literals or universally quantified formulas of the form $\forall x_1 \dots \forall x_n \,.\, (\ell_1 \wedge \dots \wedge \ell_k \rightarrow \ell_{k+1})$, where $\ell_1, \dots, \ell_{k+1}$ are literals. It can express a variety of authorization policies used in practice and also decide authorization in polynomial time. In contrast to our framework, Lithium allows predicates and function symbols of any arity as long as they respect a set of conditions (see [31] for details). For example, in a Lithium policy, there must not be two formulas, which contain two literals ℓ and ℓ' such that ℓ and $\neg\ell'$ are unifiable. However, they cannot handle numeric constraints and their policy analysis framework is limited. They only show how to efficiently decide whether, for a given policy, a permission is permitted and denied at the same time.

2.7 Conclusion and future work

Many policy specification languages have been proposed and new languages are often more expressive than their predecessors. However, policy analysis can only handle a fragment of the languages for which they are intended [28, 38, 46, 58]. This is also the case for RBAC where new extensions have richer expressiveness, but policy analysis becomes more difficult.

In this work, we have presented FORBAC as a framework that strikes a balance between expressiveness and efficient policy analysis for RBAC. We have provided strong evidence that FORBAC can specify and reason about relevant policies. As future work, we propose to extend FORBAC with two RBAC idioms that have received attention in the literature. The first is role hierarchies, which define a partial order on a set of roles. Roles are assigned those permissions granted to that role and those granted to subroles in the hierarchy. There is no standard that specifies how to define role hierarchies with role templates. The second idiom consists of constraints. The NIST standard for RBAC [26] offers two types of constraints: static and dynamic separation-of-duty constraints and cardinality constraints. It remains open how to define these constraints in RBAC models where users, roles, and permissions have attributes.

Chapter 3
N-Tube: Neighbor-Based Tube-Fair Bandwidth Reservation

3.1 Introduction

Guaranteed protection against DDoS attacks remains an open research problem. The increasing sophistication of attacks has not yet been neutralized by progress in scalable, cost-effective defenses. In sophisticated attacks, the attacker does not target the victim directly, but only a few critical links of the network that carry the victim's traffic. For example, in Crossfire, a botnet sends low-volume flows to public servers that are chosen to flood critical links required for the victim's traffic [44]. Similarly, in the Coremelt attack, the adversary sets up traffic flows among pairs of bots that it controls in a way that floods critical links [63]. For such attacks, an attacker with limited resources can effectively attack critical links and degrade connectivity for large Internet regions [43]. Defending against such attacks is currently impossible since the congestion hotspots may be far from the victim, thus removing any possible recourse.

DDOS protection is related to an effective quality of service (QoS) scheme that can provide hard guarantees in the face of strong adversaries. Since best-effort delivery and over-provisioned network bandwidth enable good performance in the average case, offering QoS guarantees requires fair resource allocation when bandwidth becomes scarce [60]. Previous QoS architectures, such as IntServ and DiffServ, were designed for an Internet with trusted network participants, not for adversarial scenarios [15, 17]. It remains an open research problem how to allocate bandwidth in an adversarial context such that legitimate hosts obtain useful bandwidth guarantees.

A core challenge is that current link-flooding attacks can be caused by a huge number of low-volume flows originating from colluding legitimate-looking bots, e.g., as seen in the Hidden Cobra DDoS Botnet Infrastructure [66]. Therefore, commonly known fairness notions that QoS solutions try to achieve, such as per source [53], per destination [71], per flow [22], per computation [56], and per class [35], are insufficient in such settings and result in an unfair sharing of bandwidth. These fairness notions suffer from the "tragedy of the commons" [32], whereby the incentive of rational agents to increase their share of a commonly available resources will lead to infinitesimally small shares for less aggressive, honest agents. In particular, in today's Internet, TCP fairness is the most commonly used *per-flow fairness* notion, which allows adversarial agents to request arbitrarily many flows and thereby obtain a disproportional amount of bandwidth compared to honest agents [18].

To address the above challenge, we design a bandwidth reservation algorithm with the following goals: (1) availability: any reservation request will promptly reserve bandwidth, (2) immutability: reserved bandwidth is guaranteed until expiry, (3) stability: the allocated bandwidths stabilize quickly if constantly renewed, (4) minimum bandwidth guarantee: there is a lower bound on the allocated bandwidth, and (5) fairness: bandwidth is fairly partitioned among the requesting parties. Our algorithm should provably satisfy these goals, even in the presence of congestion (such as during link-flooding attacks). Moreover, it should be efficient and scalable.

To achieve these goals, we focus on networks that support *path-based forwarding*, where the sender chooses an unidirectional path along which data packets are forwarded, which is recorded in the data packets' headers. New Internet architectures, such as NIRA [70], SCION [57], and Pathlets [30], are prominent examples of such architectures and indicate that path-based forwarding is feasible in large-scale networks like the Internet. Path-based forwarding plays a key role in our work: it ensures that the paths taken by data packets stay fixed and correspond to the reserved paths. In contrast, today's Internet uses forwarding tables that are frequently updated, which greatly complicates reasoning about resource allocation.

Our algorithm, called the Neighbor-based Tube-fair bandwidth reservation algorithm (N-Tube), allows autonomous systems (ASes) to reserve bandwidth on network paths. To allocate bandwidth on a path, each AS on the path computes and allocates bandwidth locally while accounting for other reservations. N-Tube builds on two key ideas:

First, to enable the immediate allocation of bandwidth (1), N-Tube only uses a fixed fraction $\delta < 1$ of the available bandwidth, saving the rest for future requests. Together with the immutability (2) of reservations until their expiry or deletion, this enables a quick stabilization (3) of the bandwidth allocations if a fixed set of reservations is continuously renewed.

Second, N-Tube is based on a novel fairness notion, called *bounded-tube fairness*, where each agent's aggregated bandwidth demands are first *limited* by the available bandwidth resources and then split proportionally among its immediate network neighbors. Hence, if a malicious AS (outside the path) tries to congest a link, the first AS between the malicious AS and the attacked link limits the adversarial AS' demands and thereby prevents it from obtaining a disproportional share of bandwidth on the attacked link. We show that our algorithm stabilizes in an ideal bandwidth allocation, which is bounded-tube fair (5) and provides a guaranteed lower bound on the allocated bandwidth (4) to the source AS, independently of the destination it wishes to reach. This holds even in the presence of adversarial demand bursts, including link-flooding attacks such as Crossfire or Coremelt.

Contributions. The main contribution of this work is the first principled solution to the global bandwidth allocation problem in adversarial networks: We provide formalizations of the N-Tube algorithm including a strong attacker model and the security goals (1)-(5) above, and we prove that our model satisfies these properties. To the best of our knowledge, this is the first bandwidth reservation algorithm that simultaneously offers properties (3)-(5) in an adversarial setting. We consider this a significant step towards a new era of DDoS-free networking.

3.2 Background

In this section we provide a high-level account of the main properties a bandwidth reservation algorithm should satisfy as well as assumptions on the network and the attacker. We will formalize these assumptions and properties in Sections 3.5 and 3.6 and sketch the correctness proofs in Section 3.6. The full proofs are given in Appendix B.

3.2.1 Goal and Associated Properties

Our goal is to design a provably secure bandwidth reservation architecture that provides hard, worst-case bandwidth guarantees to source ASes for reaching their destination ASes. A key component of such a QoS architecture, and our focus in this work, is a *bandwidth reservation mechanism* that allocates bandwidth according to the demands of source ASes and guarantees that a minimum bandwidth is allocated even under heavy congestion or flooding attacks. We have designed our bandwidth reservation algorithm N-Tube to provide the following properties.

P1 **Availability**: Any reservation request can immediately reserve bandwidth, even if there is network congestion.

P2 **Immutability**: The allocated bandwidth of any existing reservation stays fixed until the reservation expires.

P3 **Stability**: In periods of steady and constant demand, the bandwidth allocation in the entire network stabilizes quickly.

P4 **Minimum Bandwidth Guarantee**: Shortly after a reservation is set up, there is a lower bound on the allocated bandwidth, i.e., a minimal bandwidth guarantee even in the presence of high external demands like link-flooding attacks.

P5 **Bounded Tube Fairness**: Bandwidth allocation is distributed proportionally to the requested demands, however, adjusted to the maximally available bandwidth.

Note that when designing our algorithm, we also accounted for the following additional requirements. N-Tube should be *efficient* in computing bandwidth allocations using only local information of the network, i.e., based on the demands of neighbor ASes. N-Tube should be *scalable* by reducing its configuration effort for AS administrators managing bandwidth reservations, since bandwidth allocations are computed automatically. However, N-Tube allows AS administrators to specify bandwidth restrictions between their neighbor ASes to adjust minimum bandwidth guarantees. Finally, N-Tube should provide ASes with *flexibility* by allowing them to reserve segments of paths, and to update and delete reservations.

3.2.2 Model and Assumptions

Network Model. We model the network as a connected graph with weighted, directed edges. Nodes in the graph represent the AS's network interfaces and the directed edges represent physical links. Each link starts at an egress interface of an AS, called an *egress link* of this AS, and ends at an ingress interface of another AS, called an *ingress link* of that AS, and has a weight corresponding to its capacity. Using interfaces instead of multiple edges between ASes provides a simple graph, instead of an equivalent multigraph, network model, which is closer to N-Tube's implementation. Furthermore, we assume an inter-domain control plane that enables each AS to obtain multiple loop-free paths to reach a destination AS. These paths are expressed at the granularity of interfaces between the ASes. Finally, we assume that there is a globally, loosely synchronized time, i.e., with a time discrepancy between ASes on the order of seconds, in contrast to reservation times on the order of minutes.

Attacker Model. Any AS outside of the path of a legitimate reservation request may be compromised or malicious, e.g., part of a botnet. In particular, there are no constraints on the distribution of compromised ASes. We consider malicious ASes that (possibly) collude and attempt to allocate excessive amounts of bandwidth in order to exhaust the available bandwidth to be shared with other ASes. More specifically, ASes can attempt to request excessive bandwidth through (multiple) reservations over one or more paths. Moreover, we assume that senders authenticate all fields in their reservation requests and that attackers cannot defeat cryptography, e.g., spoof a signed message without the appropriate private key. Hence, attackers can replay legitimate reservation requests, but cannot craft new ones for ASes they do not control.

Attacks launched by routers located inside ASes on the path that intentionally modify, delay, or drop traffic (beyond the natural drop rate) are out of this work's scope. Such attackers could execute DoS attacks by ignoring reservations and blocking any data traffic directly. We also do not consider data-plane attacks in which the adversary sends excessive traffic or excessive amounts of bandwidth requests. In this work we design a secure bandwidth reservation algorithm, thus data-plane attacks are out of scope.

3.3 N-Tube overview

We want to enable ASes to reserve bandwidth on network paths by reserving bandwidth on each inter-domain link of a path. A reservation consists of a path, an expiration time, and an amount of bandwidth. The reservation is valid for a limited time period after which it must be renewed. This allows ASes to probe the network for congestion and adjust their reservation paths and demands.

To reserve bandwidth, the source AS chooses a path, an amount of bandwidth and an expiration time, combines them in a reservation message, and authenticates them, e.g., with RPKI [51]. In Figure 3.1, we illustrate the reservation process. The source AS_1 sends a reservation message asking for `60 Gbps` valid until `15:50:45` on the path given by the list of ASes and their corresponding ingress and egress interfaces $[(E,AS_1,C),(B,AS_2,A),(C,AS_3,D)]$. On the way to AS_3, the reservation message accumulates the amount of bandwidth each AS on the path can allocate on its egress link associated to the path: `60 Gbps`, `37 Gbps`, and `45 Gbps`, respectively. On the return path, each AS allocates the minimum, which is `37 Gbps`.

N-Tube uses two parameters that are set by its users. First, N-Tube enforces an upper bound on how long the expiration time can be set into the future, given by the parameter $maxT \in \mathbb{N}$. This forces ASes to update their reservations regularly, about every 5 minutes. Second, N-Tube only reserves a fixed portion of each link's total capacity, which is called the

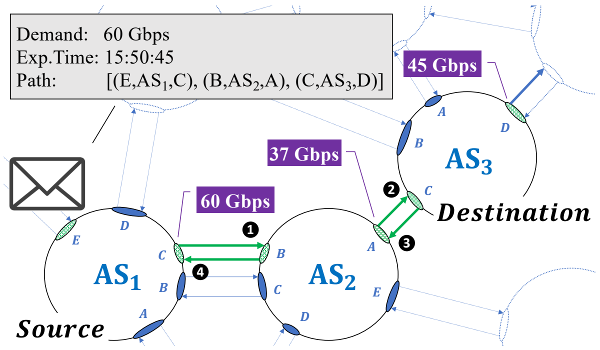


Figure 3.1: The process of making a reservation.

adjusted capacity and is given by the parameter δ, where $0 < \delta < 1$. For any new reservation request, N-Tube again initially allocates at most the portion δ of the remaining free capacity and thereby keeps the rest of the link's capacity available for other new reservations.

3.3.1 Bounded Tube-Fairness

The main challenge for a resource allocation algorithm is to treat all reservations fairly and to provide a *minimum bandwidth guarantee* for honest ASes, even when adversarial ASes try to congest a link by demanding as much bandwidth on it as possible. To provide *fair* bandwidth allocation, we bound excessive demands by the links' capacities and share the resulting demands proportionally.

The two properties of *bounded tube-fairness* and *minimum bandwidth guarantee* are achieved by N-Tube's bandwidth allocation computation. The main idea is illustrated in Figure 3.2 for AS_3 in the example given above:

1. AS_3 factors the demands going to a given egress interface by each ingress interface. These factored demands are called *tubes*.

2. AS_3 limits the accumulated demands of each tube by its ingress and egress links' adjusted capacities, which we call *bounded tube demands*.

3. AS_3 proportionally shares the egress link's adjusted capacity between its bounded tube demands.

AS_3 has three interfaces A, B, and C with ingress link capacities `100 Gbps`, `200 Gbps`, and `125 Gbps`, and and an interface D with an egress link capacity of `150 Gbps`, respectively. The link's adjusted capacities are obtained by multiplying each link's capacity with $\delta = 0.8$ and are indicated by the dotted lines. The three demands of `20 Gbps`, `80 Gbps`, and `60 Gbps` coming from interfaces A, B, and C are factored into three tubes and the adjusted capacity of `120 Gbps` at interface D is proportionally split among them into `15 Gbps` for A, `60 Gbps` for B, and `45 Gbps` for C.

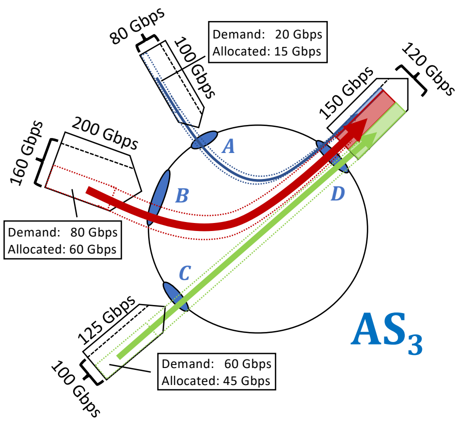


Figure 3.2: N-Tube's bandwidth allocation computation at AS_3 distributes the egress link's (adjusted) bandwidth capacity D proportionally to three ingress demands.

3.3.2 Minimum Bandwidth Guarantee

By bounding the accumulated demands of tubes in the second step of the bandwidth allocation computation, we can guarantee that each tube gets a fair share of the egress link's capacity. Whenever we must reduce a tube, we say it has *excessive demands* and we proportionally reduce all of the demands inside it.

We illustrate the main ideas of N-Tube's bandwidth allocation computation in three adversarial examples for AS_3. We assume that all ASes on the given path are honest and any AS outside of this path may be adversarial. The goal of the adversarial ASes is to reduce as much as possible the allocated bandwidth for the use by honest ASes between the interface C and the interface D. Therefore, we allow adversarial ASes to demand an arbitrary amount of bandwidth to subsequently congest the egress link at interface D. We then show how the bandwidth allocation computation still provides a minimum bandwidth guarantee.

Limit demands on an ingress link by its adjusted capacity: In the example in Figure 3.3, two adversarial ASes demand in total `400 Gbps` of bandwidth trough interface B to D. N-Tube bounds these demands by the ingress link's adjusted capacity at interface B of `160 Gbps`. Hence, D's adjusted capacity of `120 Gbps` is split proportionally between `20 Gbps` from A, `60 Gbps` from C, and `160 Gbps`, instead of `400 Gbps`, from interface B.

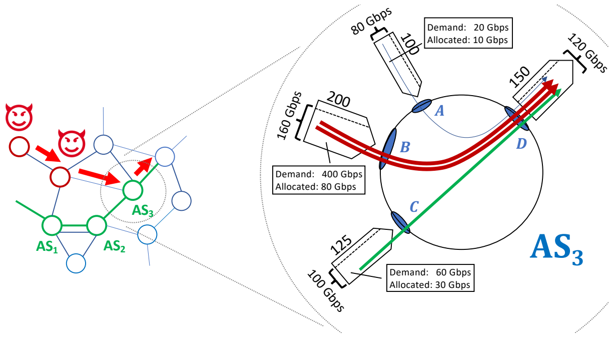


Figure 3.3: Limit demands on the ingress interface.

Limit each AS's demands by the egress link's capacity: In the example in Figure 3.4, an adversarial AS demands in total `200 Gbps` to interface D, by asking for `80 Gbps` and `120 Gbps` through A and B, respectively. However, since its combined demand of `200 Gbps` still exceeds the egress link's capacity, N-Tube reduces both demands proportionally by multiplying them with a *scaling factor*. The scaling factor is the ratio of the egress link's adjusted capacity of `120 Gbps` and the total adjusted demand of the adversarial AS of `200 Gbps`, i.e., `120/200` = 0.6. This results in the allocated demands of `48 Gbps` and `72 Gbps` from interfaces A and B, respectively. Note that, in this case, each AS must keep per-source AS state, i.e., how much bandwidth each source AS has reserved through this AS. However, this is feasible since the number of ASes in a network is much smaller than the number of flows. Hence, D's adjusted capacity of `120 Gbps` is split proportionally between `60 Gbps` from interfaces C, and the reduced demands of `48 Gbps` and `72 Gbps`, from interfaces A and B.

Worst case, minimum bandwidth guarantees: In this example, given in Figure 3.5, all ASes outside the path are adversarial and demand as much as they can on all ingress links, i.e., a maximum of `80 Gbps` on interface A and `160 Gbps` (i.e., `80 Gbps` each) on interface B. Note that this represents

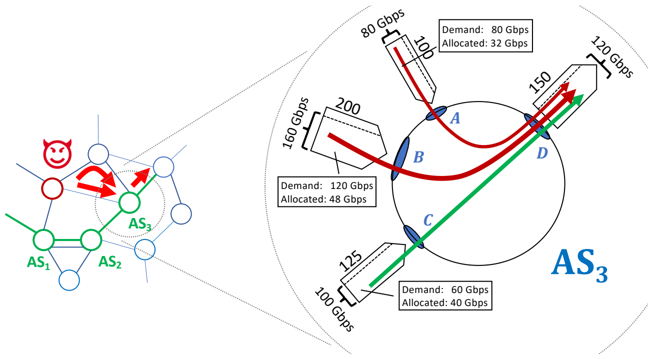


Figure 3.4: Limit demands on the egress interface.

a worst-case attack since even when more adversarial ASes are present their bandwidth demands will be adjusted and are therefore limited as described in the two previous examples. The interface D's adjusted capacity of `120 Gbps` is split proportionally between the bounded demands of `80 Gbps` from interface A and `160 Gbps` from interface B and the honest demand of `60 Gbps` from interface C. Hence, this honest demand cannot be reduced to less than `24 Gbps` by any amount of external demands. This provides the minimum bandwidth guarantee for this reservation at AS_3.

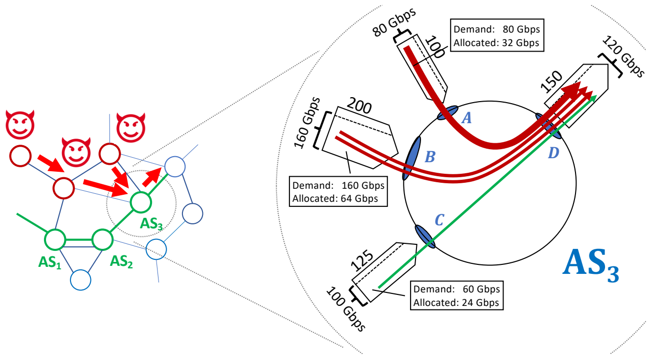


Figure 3.5: Minimum bandwidth guarantee in the worst case.

By applying the idea from the last example to each of the path's ASes, one can show that N-Tube's bandwidth allocation computation provides *minimum bandwidth guarantee* (P4). Furthermore, N-Tube's computation splits the adjusted link's capacity proportionally between the non-excessive demands. This illustrates, informally, that N-Tube's bandwidth allocation computation provides the property of *bounded tube-fairness* (P5) for each link.

3.3.3 Additional properties

By only reserving a proportion given by δ of the available bandwidth, N-Tube can always provide a positive (but possibly small) amount of bandwidth for new reservations. This ensures *availability* (P1) and is proven in Section 3.6. To optimize the links' utilization, unused bandwidth capacities are provided for best-effort traffic.

By N-Tube's design, the only two ways for a reservation to become invalid is either for it to expire, when it is not successfully renewed, or to be deleted by a deletion message sent by the source, as described in Section 3.4. Otherwise N-Tube does not modify existing reservations. This results in the property of *immutability* (P2), which we also prove in Section 3.6.

The upper bound *maxT* on how long the expiration time can be set into the future forces ASes to renew their reservations regularly. This allows N-Tube to stabilize the allocations quickly after a burst in demands, i.e., the *stability* property (P3). We prove in Section 3.6 that after a $2maxT$-period of stable demands, N-Tube's computations converges to a stable state of allocations, which satisfies our notion of *bounded tube-fairness*.

Finally, we argue that allowing ASes to reserve path segments along paths and renewing and deleting reservations provides sufficient *flexibility* for source ASes to adapt their reservations. Furthermore, since the N-Tube bandwidth allocation computation for each reservation is done locally, and only involves simple arithmetic computations, each AS can execute the reservation's bandwidth allocation computation *efficiently* in the number of already existing reservations. Additionally, N-Tube minimizes its communication complexity by requiring only one round-trip per request.

3.4 Algorithm details

After introducing some notation, we define the network model, messages and reservation maps. Afterwards we describe how our distributed bandwidth allocation algorithm N-Tube processes messages. In the last subsection we specify N-Tube's bandwidth allocation computation in detail and describe the local properties it achieves.

3.4.1 Notation

As *base types*, $\mathbb{1} = \{\perp\}$ denotes the unit type, $\mathbb{B} = \{\text{TRUE}, \text{FALSE}\}$ denotes the booleans, $\mathbb{N} = \{0, 1, 2, \dots\}$ denotes the natural numbers, and $\mathbb{Q}_0^+$ and

$\mathbb{R}_0^+$ denote the non-negative rational and the non-negative real numbers, respectively. Given elements a and b of the same numeric type, with $a \leq b$, we denote open and closed intervals by $]a; b[$ and $[a; b]$, respectively.

Given two types A and B, we denote the *function type* with *domain* A and *co-domain* B by $A \to B$, their *product type* by $A \times B$, and their *sum type* by $A + B$. We define partial functions $A \rightharpoonup B = A \to B_\perp$ with $B_\perp = B + \mathbb{1}$, the *support* of a partial function g by $\mathit{supp}(g) = \{a \in A \mid g(a) \neq \perp\}$, and its *range* by $\mathit{rng}(g) = \{b \in B \mid \exists a \in A.\ g(a) = b\}$. In case $g(x)$ appears in a predicate P we implicitly assume $P(g(x)) \wedge g(x) \neq \perp$. We denote partial functions with finite support by $A \rightharpoonup_f B$ and the undefined function with empty support by $\emptyset$.

We identify types with sets and also use standard set notation. For example, for a set A we write $\mathcal{P}(A)$ for its *power set*, $\mathcal{P}_f(A)$ for the set of its finite subsets, $|A|$ for its *cardinality*, A^n for its *n-ary product*, A^* and A^ω for its *finite* and *infinite sequences*, and $A^\infty := A^* \cup A^\omega$ for its *sequences*.

By $f(x \mapsto y)$, we denote the function that behaves like f except that it maps x to y, i.e.,

$$f(x \mapsto y)(x') := \begin{cases} y, & \text{if } x' = x \\ f(x'), & \text{otherwise.} \end{cases}$$

Informally, a *record type* is a product type together with names for its projections, as shown in the following example: $\mathit{point} = (\!|\ x \in \mathbb{N}; y \in \mathbb{N}\ |\!)$ with elements like $p \equiv (\!|\ x = 1; y = 2\ |\!)$ and projections $p.x$ and $p.y$. The term $p(\!|\ x := 3\ |\!)$ denotes the updated point $(\!|\ x = 3; y = 2\ |\!)$. The type $\mathit{cpoint} = \mathit{point} \oplus (\!|\ c \in \mathit{color}\ |\!)$ extends *point* with a *color* field.

The inductive type of *lists* of type A is defined as $[A] = \mathbb{1} + A \times [A]$, with the constructors $\mathit{nil} : \mathbb{1} \to [A]$ providing the empty list $\mathit{nil}(\perp)$ and $\# : A \times [A] \to [A]$ for prepending an element $a \in A$ to a list $l \in [A]$. To simplify notation, we write *nil* instead of $\mathit{nil}(\perp)$ and $a\#l$ instead of $\#(a, l)$. We also use conventional list notation , e.g., $[1, 2, 3]$[1] for the list $1\#2\#3\#\mathit{nil}$ and for a list $l \in [A]$ and $n \in \mathbb{N}$ we write $l[n]$ to retrieve the n's element in l, starting from $n = 0$. The functions min and max respectively yield the minimum and maximum element of a non-empty finite set of numbers and $+\infty$ and 0 for the empty set. We extend them to tuples, lists, and records of numbers (by taking the set of their components) and to partial functions with finite support and numerical co-domain (by taking the range).

[1]Note the syntactic difference between the closed interval $[1; 3]$, the pair $(1, 3)$, and the two element list $[1, 3]$.

3.4.2 Message Processing

Network We model the *network* as a weighted, directed graph (N, E, cap) for which we give a simplified definition here. Full details are given in Appendix A.1.

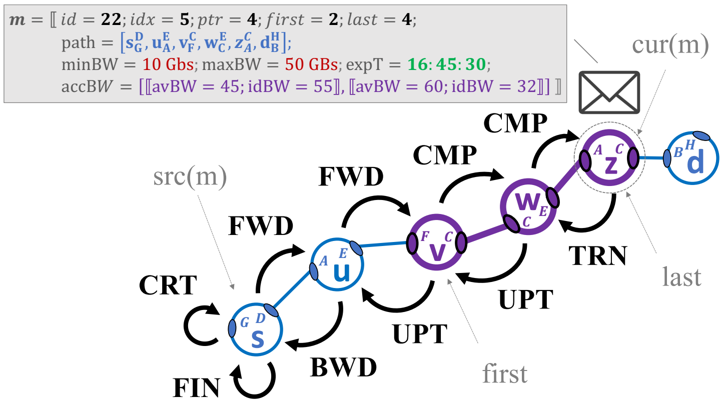
- The nodes N are given by the set $V \times I$, where V is a finite set of vertices and I provides a set of identifiers for interfaces inside of each AS.
- The finite set of directed edges $E \subseteq N \times N$ represents the physical *links* between ASes. Given a link $((u,e),(v,i)) \in E$ we call e its *egress* interface at AS u and i its *ingress* interface at AS v. We assume that at any AS interface there is either exactly one *ingress* and one *egress* link or no link at all.
- The *capacity* of each link is given by the non-negative real-valued function $cap : E \to \mathbb{R}_0^+$. Since a link $l = ((u,e),(v,i))$ is uniquely defined by (u,e) (and (v,i)), we identify $cap(l)$ with $cap(u,e)$ (and $cap(v,i)$).

We define the type of *paths* $\mathcal{P}$ as the lists of records with ingress interface inI, AS identifier as, and egress interface egI:

$$\mathcal{P} = \left[\llparenthesis\, inI \in I; as \in V; egI \in I \,\rrparenthesis \right].$$

Given a network, we call a path $p \in \mathcal{P}$ *valid*, if (i) it is non-empty, (ii) each edge corresponding to p's ingress and egress interfaces is an inter-AS link, i.e., it starts and ends in distinct ASes, and (iii) its set of links is *connected* and *directed*, i.e., the edges connect the ASes in p and they all point in the same direction, and (iv) *loop-free*, i.e., that each AS occurs at most once on the path. In the following, we denote elements of the sets V and I with lowercase letters, specifically for V we use the letters s, x, v, and z, for ingress interfaces i and i', and for egress interfaces e and e'. To simplify notation, we write the ingress interface as subscripts and the egress interface as superscript to the AS identifier.

Messages There are two types of messages: *reservation* and *deletion messages*. The fields of a message identify the reservation, state how the message should be routed through the network, how much bandwidth should be reserved, and until when the reservation should be valid. We introduce the fields of a reservation message m in Figure 3.6:



$\boldsymbol{m} = [\![\, id = \mathbf{22}; idx = \mathbf{5}; ptr = \mathbf{4}; first = \mathbf{2}; last = \mathbf{4};$
$\quad \text{path} = [\mathbf{s}^{\mathbf{D}}_{\mathbf{G}}, \mathbf{u}^{\mathbf{E}}_{\mathbf{A}}, \mathbf{v}^{\mathbf{C}}_{\mathbf{F}}, \mathbf{w}^{\mathbf{E}}_{\mathbf{C}}, \boldsymbol{z}^{\boldsymbol{C}}_{\boldsymbol{A}}, \mathbf{d}^{\mathbf{H}}_{\mathbf{B}}];$
$\quad \text{minBW} = 10\ \text{Gbs}; \text{maxBW} = 50\ \text{GBs}; \text{expT} = \mathbf{16{:}\,45{:}\,30};$
$\quad \text{accB}W = [[\![\text{avBW} = 45; \text{idBW} = 55]\!], [\![\text{avBW} = 60; \text{idBW} = 32]\!]]\ ]\!]$

Figure 3.6: The message m is processed along p by the events described in Section 3.5.3.

- The source $s \in V$ of the path (see below) can choose any reservation ID $id \in \mathbb{N}$ $(= 22)$. The pair $(s, 22)$ of source identifier and reservation ID uniquely determines a reservation in the network. Furthermore, the source provides an index idx $(= 5)$ that indicates which version of the reservation the messages refers to. Version indices are used to update already existing reservations and are explained later.

- The field $path \in \mathcal{P}$ $(= p)$ provides the path and the field $ptr \in \mathbb{N}$ $(= 4)$ provides the pointer to the AS $(p[4].as = z)$, where the message is currently processed. The fields $first \in \mathbb{N}$ $(= 2)$ and $last \in \mathbb{N}$ $(= 4)$ point to the first and last AS on p, respectively.

- The minimum $minBW \in \mathbb{R}^+_0$ $(= 10\ \text{GB/sec})$ and maximum bandwidth $maxBW \in \mathbb{R}^+_0$ $(= 50\ \text{GB/sec})$ state the range of bandwidth the source AS would like to reserve. Hereby, $maxBW$ states the source's *demand* in this reservation request. In case the source's demand cannot be provided on p, $minBW$ states the *minimal* amount of bandwidth the source is willing to accept as a reservation.

- The expiration time $expT \in \mathbb{N}$ $(= 16{:}45{:}30)$ indicates when the reservation expires and must be deleted by the ASes on the path p.

- The list of bandwidth values $accBW \in [(\!|\ avBW, idBW \in \mathbb{R}^+_0\ |\!)]$ indicates the *available* and *ideal* bandwidth the previous ASes (v and w in the example) were able to provide, as explained in Section 3.4.3.

The functions *src, first, cur,* and *sgmt* on valid messages respectively extract from *m.path* the *source AS,* the *first AS,* the *current* AS with its ingress and egress interfaces, and the *set of ASes between first and last.* In the example of Figure 3.6 $src(m) = s$, $first(m) = v$, $cur(m) = z_A^C$, and $sgmt(m) = \{v, w, z\}$. In a deletion message, the fields *first, last, minBW, maxBW, expT,* and *accBW* are omitted.

Formally, the type of *messages* $\mathcal{M}$ is the sum-type of *reservation messages* $\mathcal{M}_R$ and *deletion messages* $\mathcal{M}_D$

$$\mathcal{M} = \mathcal{M}_D + \mathcal{M}_R$$

with

$$\begin{aligned}
\mathcal{M}_D &= (\!|\, id, idx \in \mathbb{N}; path \in \mathcal{P}; ptr \in \mathbb{N} \,|\!) \\
\mathcal{M}_R &= \mathcal{M}_D \oplus (\!|\, first, last \in \mathbb{N}; \\
&\qquad\qquad minBW, maxBW \in \mathbb{R}_0^+; expT \in \mathbb{N}; \\
&\qquad\qquad accBW \in [\, (\!|\, avBW, idBW \in \mathbb{R}_0^+ \,|\!)\,] \,|\!)
\end{aligned}$$

A message *m* is *valid,* if it contains a valid path in its field *m.path* and for its pointers it holds that $0 \le ptr, first, last \le length(m.path)$ and $first < last$. The bandwidth range must be a non-empty interval, i.e., $0 \le m.minBW \le m.maxBW$ and $m.maxBW > 0$, hence only non-zero bandwidth allocations are allowed.

Reservation maps Each AS maintains its own reservation map where it stores all currently valid reservations with a path traversing it. A reservation map is a partial function that maps a source and a reservation ID to a record containing the following fields: The reservation's path *path,* the pointers *ptr, first* and *last*, and a *version map vrs.* For example in the reservation map $resM_z$ of AS *z*, the entry *rs* corresponding to message *m* from Figure 3.6 is

$$resM_z(s, 22) = (\!|\, path = p; ptr = 4; first = 2; last = 4; vrs = verM \,|\!).$$

To provide flexibility, N-Tube allows source ASes to update their reservations multiple times before they expire and therefore stores different versions of each reservation in the *version map* field *vrs*.

A version map is a partial function that maps each reservation index to a record containing the following fields: The minimal bandwidth *minBW,* the maximal bandwidth *maxBW,* the ideal bandwidth *idBW* computed by the previous AS *w,* the expiration time *expT* given by *m,* and the reserved bandwidth *resBW* determined by the N-Tube algorithm. We call the entries in the version map *versions* of its corresponding reservation, e.g., for m

$$\begin{aligned} verM(5) = \llparenthesis\ & minBW = 10\ \text{GB/s}; maxBW = 50\ \text{GB/s}; \\ & idBW = 60\ \text{GB/s}; resBW = 32\ \text{GB/s}; expT = 16{:}45{:}30\ \rrparenthesis. \end{aligned}$$

A version vr is *currently valid* at time t, written $cvalid(vr, t)$, if it is *not expired*, i.e., $vr.expT \geq t$, and *successful*, i.e., $vr.minBW \leq vr.resBW$. The reservation's bandwidth *demand* and *allocation*, *demBW* and *allocBW*, are defined as the maximum of its currently valid versions' *maxBW* and *resBW*, respectively.

$$demBW(rs, t) = \max_{vr \in rng(rs.vrs)} \{vr.maxBW \mid cvalid(vr, t)\}$$
$$allocBW(rs, t) = \max_{vr \in rng(rs.vrs)} \{vr.resBW \mid cvalid(vr, t)\}$$

The maximum is taken, since the source can send traffic using any existing version of its reservations. Hence, this computation guarantees that, in the worst-case, enough bandwidth is available.

Reservation process The N-Tube algorithm processes reservation messages depending on their direction and position on the path as shown in Figure 3.6. If AS s wants to make a new reservation *id* on a path p, it creates (CRT) a reservation message m containing p in its *path* field. The ASes located before *first* on p just forward (FWD) the message along p by increasing m's *ptr* field. If m reaches the ASes between *first* and *last*, each AS x processes (CMP) m by:

1. checking that $resM_x$ does not contain a reservation at (s, id) or there is a reservation at (s, id) for the same path but no valid version map entry at *idx*,
2. computing how much bandwidth is *available* at x and how much x can *ideally* provide for the reservation (details of the computation will be given shortly in Section 3.4.3),
3. updating m to a new message m' by appending the computation results to the *accBW* and incrementing *ptr*,
4. sending m' to the next AS on the path p, and
5. adding a new version at index *idx* of the reservation identified by (s, id) in $resM_x$.

After the last AS z, given by the pointer *last*, on the path p has processed m, it sends the message m' backward (TRN) on the path. During the backward traversal, each of the path's ASes extracts how much bandwidth *finBW* could be reserved on the entire path by taking the minimum of m's fields *maxBW* and *accBW*, i.e., what has been computed by each AS on the forward traversal

$$\mathit{finBW}(m) = \min(m.\mathit{maxBW}, \min(m.\mathit{accBW}))\,.$$

Analogously to the forward traversal updates, the AS updates (UPT) its reservation map according to the same two cases: Either (i) the reservation was successful, i.e., $m.\mathit{minBW} \leq \mathit{finBW}$, and each AS on the path updates the reserved bandwidth of the corresponding version in its reservation map to *finBW*, or (ii) there was not enough bandwidth available on the path, i.e., $m.\mathit{minBW} > \mathit{finBW}$, and each AS deletes the corresponding version from its reservation map. The ASes between *first* and the source just send the message backwards (BWD) without processing it until s receives it in the end (FIN).

Renewal If s wants to renew one of its reservations with a given ID *id*, it sends a new reservation message m containing an updated bandwidth range and expiration time, along the previous path p. The index *idx* of m must be fresh, i.e., s has not used it yet and the pointers *first* and *last* must remain the same. Otherwise the renewal is invalid and m is dropped.

Deletion If s wants to delete a version with index *idx* of one of its reservations with ID *id*, it sends a deletion message along p with the corresponding *id* and *idx*. Each AS on p processes the deletion message by removing the version entry at *idx* in the reservation identified by (s, id). In contrast to a complete round-trip, required for set-up and renewal, the last AS drops the message, i.e., only one traversal along p is required for deletion.

3.4.3 Fair Bandwidth Allocation

The heart of the N-Tube algorithm is the bandwidth allocation computation. We assume that a *valid* reservation message m was sent by its source AS s and arrives at an AS x that lies between *first* and *last* on m's path at the current time t. First, the N-Tube algorithm derives its source s $(= \mathit{src}(m))$ and its current AS, ingress, and egress interfaces x_i^e $(= \mathit{cur}(m))$. Given m and resM_x, the N-Tube computation determines:

- the *available* bandwidth, i.e., how much bandwidth remains on the link at the egress interface e, and

- the *ideal* bandwidth, i.e., how much bandwidth s is requesting compared to all active reservations in $resM_x$ from interface i to e.

These computations are given by the functions *avail* and *ideal*, and are described in detail in the following. To simplify notation, we fix the message m and its elements s, id, x, i, and e, and we omit $resM_x$, t, and the parameter δ as arguments. For a complete description, see Appendix A.9.

Available Bandwidth Computation

Given the message m, the function *avail* computes how much bandwidth is *available* on the link at the egress interface e of AS x

$$avail(e) = \delta \cdot \left(cap(x,e) - \sum_{\substack{r' \in rng(resM_x):\\ resEg(r')=e}} allocBW(r') \right).$$

This function subtracts the aggregated *allocated bandwidth* of all currently valid reservations with the same egress interface e from the link's total capacity $cap(x,e)$ and multiplies the result by the parameter δ to obtain the remaining bandwidth. Multiplying with $0 < \delta < 1$ guarantees that some bandwidth is always available for subsequent reservation requests. The functions *resSr*, *resEg* and *resIn* extract the corresponding reservation's source AS and egress and ingress interface, see Appendix A.9 for details.

Limiting Excessive Demands

To avoid that s reserves more bandwidth in one request than physically possible, we limit the bandwidth demand $demBW(r.vrs)$ of a reservation r by the ingress and egress links' capacities. The resulting *requested demand* of a reservation r is defined by

$$reqDem(r) = \min\{cap(x,i), cap(x,e), demBW(r)\}.$$

This guarantees that

$$reqDem(r) \leq \min\{cap(x,i), cap(x,e)\}.$$

As illustrated in Section 3.3, a source's aggregated demands at a given link may exceed the link's capacity, even if none of its individual requests does. We now formally define the notion of a source having excessive demands on a link and an adjusted version of the requested demand, *adjReqDem*, to account for such demands.

The *egress demand* of s on e is defined as the aggregate over its *requested demands* with egress demand e

$$egDem(s,e) = \sum_{\substack{r' \in rng(resM_x):\\ resSr(r')=s\\ resEg(r')=e}} reqDem(r').$$

Analogously, we compute the *ingress demand* on the ingress interface i, see Appendix A.9 for details.

Definition (Excessive Demands)**.** We say an AS s *has excessive demands on the egress link* e, if $egDem(s,e) > cap(x,e)$. Otherwise, we say s *has moderate demands on* e. We call an egress link e *congested* if

$$\sum_{s' \in V} egDem(s',e) > cap(x,e).$$

Analogous definitions apply to ingress links.

To account for the case where s has excessive demands on the egress link e, we adjust the requested demand of a reservation r by multiplying it with the minimum of the corresponding ingress and egress *scaling factors*, yielding the *adjusted requested demand*

$$\begin{aligned} &adjReqDem(r) = \\ &\quad \min\{inScalFctr(s,i), egScalFctr(s,e)\} \cdot reqDem(r,i,e). \end{aligned}$$

with s, i, and e the corresponding source AS, ingress and egress interface of r, respectively.

We compute the *egress scaling factor* on the egress link at e for s as the source's proportion of the total *egress demand* bounded by the egress link's capacity, given by

$$egScalFctr(s,e) = \frac{\min(cap(x,e), egDem(s,e))}{egDem(s,e)}.$$

Analogously, we compute the source's *ingress scaling factor*, see Appendix A.9 for details.

This guarantees that the *adjusted egress demand* of s on e defined as the aggregate over its adjusted requested demands

$$adjEgDem(s,e) = \sum_{\substack{r' \in rng(resM_x):\\ resSr(r')=s\\ resEg(r')=e}} adjReqDem(r')$$

is bounded by the egress link's capacity, i.e., $\mathit{adjEgDem}(s,e) \leq \mathit{cap}(x,e)$, and analogously for the *adjusted ingress demand*.

Note that we give sources who have excessive demands the benefit of the doubt. We assume that they cannot know the links' capacities in the network and unintentionally demand too much bandwidth. Hence, instead of dropping their reservation messages, we adjust their excessive demands in the *ideal* computation by limiting and scaling them as described above.

Ideal Bandwidth Computation

Given a message m, the function *ideal* computes how the adjusted capacity $\delta \cdot \mathit{cap}(x,e)$ of the egress link e of AS x is shared in a so-called *bounded tube-fair* manner among all the existing reservations (in $resM_x$) with the same egress link e

$$\begin{aligned}&\mathit{ideal}(s,id,i,e) = \\ &\mathit{reqRatio}(s,id,i,e) \cdot \mathit{linkRatio}(i,e) \cdot \mathit{tubeRatio}(i,e) \cdot \delta \cdot \mathit{cap}(x,e).\end{aligned}$$

The function *ideal*

1. proportionally splits the egress link's adjusted capacity between each ingress link by multiplying with *tubeRatio*,

2. partitions the result between reservations starting and traversing the ingress link i by multiplying with *linkRatio*, and

3. splits the result proportionally between all starting respectively traversing reservations by multiplying with *reqRatio*.

In the following, we will define these three ratios.

Tube Ratio: The *tube ratio* between an ingress interface i and an egress interface e is computed as the ratio of the *bounded tube demand* between i and e, given by $\min\{\mathit{cap}(x,i), \mathit{tubeDem}(i,e)\}$, and the aggregated bounded tube demands at e

$$\mathit{tubeRatio}(i,e) = \frac{\min\{\mathit{cap}(x,i), \mathit{tubeDem}(i,e)\}}{\sum_{i' \in I} \min\{\mathit{cap}(x,i'), \mathit{tubeDem}(i',e)\}}.$$

Taking the minimum with respect to the corresponding ingress link's capacity guarantees that its respective portion of tube demand compared to the other ingress links' tube demands is always bounded. This prevents that the bandwidth reserved for other ingress links will be reduced ad infinitum.

The *tube demand* between an ingress interface i and an egress interface e aggregates their *adjusted requested demands*

$$tubeDem(i,e) = \sum_{\substack{r' \in rng(resM_x): \\ resIn(r')=i \\ resEg(r')=e}} adjReqDem(r').$$

Link Ratio: If x is a transit AS on m's path, i.e., $m.ptr > m.first$, then the *link ratio* between an ingress interface i and an egress interface e is computed as the ratio of the *bounded transit demand* between i and e, given by $\min\{cap(x,i), transitDem(i,e)\}$, and the sum of bounded start and bounded transit demand

$$linkRatio(i,e) = \frac{\min\{cap(x,i), transitDem(i,e)\}}{\min\{cap(x,i), startDem(i,e)\} + \min\{cap(x,i), transitDem(i,e)\}}.$$

Taking the minimum with respect to ingress link's capacity guarantees that its respective portion for transit demand compared to demands of reservations starting at i is always bounded. This prevents that the bandwidth allocated for traversing reservations can be reduced ad infinitum by excessive reservations starting at link i and vice-versa. Here *transit demand* and *start demand* are similarly defined to the *tube demand*, see Appendix A.9.

Request Ratio: The *request ratio* of a reservation identified by (s, id) between i and e is the ratio between its *adjusted ideal bandwidth allocation* (provided by the predecessor on the reservation's path) and the *transit demand* between i and e:

$$reqRatio(s,id,i,e) = \frac{adjIdBW(s,id,e)}{transitDem(i,e)},$$

with *adjIdBW* similarly defined to *adjReqDem*, see Appendix A.9.

Properties of the N-Tube Computation

If the reservation message m is valid, then N-Tube's bandwidth allocation computation has the following properties.

Positivity: The functions *avail* and *ideal* always compute strictly positive values, hence, for a valid request a positive amount of bandwidth is always allocated:

$$\forall m.\, finBW(m) > 0.$$

Lower *ideal* Bound: Let m be a valid message with source s, ID id, and first AS v together with v's ingress and egress interface i_v and e_v. There is a strictly positive lower bound $G \cdot r_v$ on the *ideal* computation (even when all sources exceed their demands), where G only depends on m's path and $r_v = \mathit{reqRatio}(s, id, i_v, e_v)$ is the request ratio of m at v.

$$\forall m. \exists\, G \in]0;1]. \forall x \in \mathit{sgmt}(m).\ \mathit{ideal}(m, \mathit{resM}_x) > G \cdot r_v.$$

Bounded Tube-Proportionality: Provided that two ingress links i, i' of AS x are not congested, the *tubeRatio* computation splits the capacity of egress link e proportionally according to the tube demands of i and i' to e

$$\frac{\mathit{tubeRatio}(i,e)}{\mathit{tubeRatio}(i',e)} = \frac{\mathit{tubeDem}(i,e)}{\mathit{tubeDem}(i',e)}.$$

In case i' is congested and its tube demand to e further increases, the ratio between both tube ratios remains fixed

$$\frac{\mathit{tubeRatio}(i,e)}{\mathit{tubeRatio}(i',e)} = \frac{\mathit{tubeDem}(i,e)}{\mathit{cap}(x,i')}.$$

Per Request-Proportionality: Suppose two sources s_1 and s_2 respectively make new reservations m_1 and m_2 whose paths intersect in x. If s_1 and s_2 do not have excessive demands on e and there is no congestion on either path up to x, then their *ideal* bandwidth computations are proportional to the messages' bandwidth demands, even if e is congested

$$\frac{\mathit{ideal}(m_1, \mathit{resM}_x)}{\mathit{ideal}(m_2, \mathit{resM}_x)} = \frac{m_1.\mathit{maxBW}}{m_2.\mathit{maxBW}}.$$

These properties follow easily from the functions' definitions and proofs are given in Appendix A.9.

Manual Tube Ratio Adjustment

Often the links' capacities may not reflect the actual demands arriving at the interfaces of an AS. Network administrators know from experience what data rates they can expect between interfaces of their ASes. For instance, there might be an oversea link ending at an AS which is much larger compared to the other ingress links. The respective AS serves as transit and simply forwards the oversea traffic to a single egress interfaces with no demands on the other egress interfaces. Another example is an AS that transfers priority traffic from a hospital or a research center. Such traffic may be small compared to less urgent traffic from video streaming services coming from other interfaces, but needs a stable bandwidth allocation for fixed time periods.

To account for such cases, N-Tube's bandwidth allocation computation can be adjusted by using *limit matrices* at each AS, instead of the links' capacities.

Limit matrix: Suppose, given an AS x and the two sets $I_x, E_x \subseteq I$ of its ingress and egress interfaces, respectively. The *limit matix* of x is a function of type $I_x \times E_x \to \mathbb{R}^+$, i.e., a matrix, that defines for each pair $(i, e) \in I_x \times E_x$ its *limited tube demand*. Note that $Limit_x(i, e) > 0$.

The *limited ingress demand* of ingress link $i \in I_x$ are the aggregated of values of $Limit_x$ fixed at i

$$Limit_x(i) = \textstyle\sum_{e' \in E_x} Limit_x(i, e').$$

Analogously, we define the *limited egress demand* for an egress link $e \in E_x$, denoted by $Limit_x(e)$, by aggregating over the ingress links $i' \in I_x$.

To adapt N-Tube's bandwidth allocation computation, we simply replace $cap(x, i)$ and $cap(x, e)$ with $Limit_x(i)$ and $Limit_x(e)$, respectively.

It is easily to see that the assumption $Limit_x(i, e) > 0$ is sufficient for proving the properties given in the previous subsection. For the *lower* ideal *bound* property the lower bound G also depends on the limit matrices of the ASes on the message's path. In case of the *bounded tube-proportionality* property the capacity $cap(x, e)$ in the denominator must be $Limit_x(e)$.

3.5 Model

In this section, we first introduce labeled transitions systems as our general modeling framework. These consist of states providing snapshots of the system and events defining its transitions. Next we model the N-Tube algorithm and its networking environment as a labeled transition system. We assume that the set of ASes V is partitioned into a set of *honest* ASes H and a set of *malicious* (or compromised) ASes M. Our model includes a definition of the state space and events modeling (i) the N-Tube algorithm by describing how honest ASes process messages, and (ii) the attack capabilities of the malicious ASes.

3.5.1 Preliminaries

Our mathematical model of the system is based on a *labeled transition system* $\mathfrak{T} = (\Sigma, \Sigma_0, \Lambda, \Delta)$ with state space Σ, initial states $\Sigma_0 \subseteq \Sigma$, labels

Λ, and transition relation $\Delta \in \Lambda \to \mathcal{P}(\Sigma \times \Sigma)$. Given a labeled transition system $\mathfrak{T}$, *executions* are functions of type $\mathfrak{E} = \mathbb{N} \to \Sigma \times \Lambda$ such that any $\pi = \{(\sigma_n, \lambda_n)\}_{n \in \mathbb{N}} \in \mathfrak{E}$

- starts from an initial state $\sigma_0 \in \Sigma_0$ and
- progresses according to the transition relation Δ

$$\forall n \in \mathbb{N}.\ (\sigma_n, \sigma_{n+1}) \in \Delta(\lambda_n).$$

To specify our concrete models, we use a structured form of transition systems, where states are given by a record type and the transition relation is defined by a finite set $\mathcal{E}$ of *events*. Each event $evt \in \mathcal{E}$ is given as a function of type $D_{evt} \to \mathcal{P}(\Sigma \times \Sigma)$ and is of the general form

$$evt(\bar{x}) = \{(\sigma, \sigma') \mid G_{evt}(\bar{x}, \sigma) \wedge \sigma'.\bar{v} = \bar{f}_{evt}(\bar{x}, \sigma)\}.$$

We write $\bar{x}$ of type D_{evt} for the event's *parameters* and $\bar{v}$ for a subset of the *state variables*. Furthermore, $G_{evt}(\bar{x}, \sigma)$ is a conjunction of *guards*, and $\sigma'.\bar{v} = \bar{f}_{evt}(\bar{x}, \sigma)$ is an *action* with *update functions* $\bar{f}_{evt}$, one for each variable in $\bar{v}$. The guards define when the event is enabled and the action is syntactic sugar for $\sigma' = \sigma(\!|\ \bar{v} := \bar{f}_{evt}(\bar{x}, \sigma)\ |\!)$, i.e., the simultaneous assignment of values $\bar{f}_{evt}(\bar{x}, \sigma)$ of variables $\bar{v}$ in state σ.
The set of labels Λ is defined by

$$\Lambda = \biguplus_{evt \in \mathcal{E}} \{evt\} \times D_{evt},$$

and the labeled transition relation Δ is defined by

$$\Delta(evt, \bar{d}) = \bigcup_{\bar{d} \in D_{evt}} evt(\bar{d}).$$

The concrete parameters $\bar{d}$ can be thought of as being chosen non-deterministically by the environment.

In the following sections, we fix the system's environment, i.e., the network graph (N, E, cap) as described in Section 3.4, a partition $V = H \uplus M$, and the fraction $0 < \delta < 1$ of the links' adjusted capacities.

3.5.2 States

The set of *states* Σ is defined as the record type

$$\Sigma = (\!|\ time \in \mathbb{N}; buf \in Buff; res \in ResMap; kwl \in \mathcal{P}(\mathcal{M})\ |\!)$$

A *state* $\sigma \in \Sigma$ describes a snapshot of the system at a given point in time, given by its *time* field. In our system model, we assume discrete time, which is *loosely* synchronized between all ASes in the network, i.e., compared to the minimal duration of reservations (on the order of minutes), the discrepancy of time measurements between AS (on the order of seconds) is negligible. The *buffers* in the field *buf* model the traversal of messages through the network as a partial function of type

$$\mathit{Buff} = V \times I \to \mathcal{P}_f(\mathcal{M}).$$

A function $\mathit{buf} \in \mathit{Buff}$ describes for every AS $x \in V$ and interface $i \in I$ the set of messages $\mathit{buf}(x, i)$ that has arrived at that interface of the AS. The field *res* models all *reservation maps* in the network are the partial functions

$$\mathit{ResMap} = V \times V \times \mathbb{N} \rightharpoonup_f \mathit{Res}.$$

A partial function $\mathit{res} \in \mathit{ResMap}$ stores for each AS x its *reservations* $\mathit{res}(x, s, \mathit{id})$ given by the corresponding pair of reservation identifiers (s, id). Note that, compared to the previous section, we combine all reservation maps resM_x into a global one, but continue using the indexed notation. Each reservation is given by

$$\mathit{Res} = (\!|\ \mathit{path} \in \mathcal{P}; \mathit{ptr}, \mathit{first}, \mathit{last} \in \mathbb{N}; \mathit{vrs} \in \mathit{VrsMap}\ |\!),$$

containing a *version map* of the partial function type

$$\mathit{VrsMap} = \mathbb{N} \rightharpoonup_f (\!|\ \mathit{minBW}, \mathit{maxBW}, \mathit{idBW}, \mathit{resBW} \in \mathbb{R}_0^+; \mathit{expT} \in \mathbb{N}\ |\!)$$

as described in the previous section.

The field *kwl* models the *attackers' knowledge* as the set of messages that malicious ASes can create or collect over time. These messages then can be modified and sent to honest ASes to execute attacks, e.g., by reserving excessive bandwidth in the network.

The set of *initial states* $\Sigma_0 = \{\sigma_0\}$ is given by the single state

$$\sigma_0 = (\!|\ \mathit{time} := 0; \mathit{buf} := \emptyset; \mathit{res} := \emptyset; \mathit{kwl} := \mathit{kwl}_0\ |\!).$$

where time starts at 0, the buffers and reservation map are initially empty, and the attackers' initial knowledge $\mathit{kwl}_0 = \{m \in \mathcal{M} \mid \mathit{src}(m) \in M\}$ consists of all messages with a malicious source AS.

3.5.3 Events

The set of events $\mathcal{E}$ can be partitioned into two kinds of events:

1. Events of the N-Tube algorithm. These consist of the N-Tube *message processing* events and an internal event that *removes expired reservations* in each AS. There are different message processing events depending on a message's type, its location on the path, and the direction of the path traversal (cf. Figure 3.6). This results in seven events describing how honest ASes process reservation messages, three events to process deletion messages, and one event for dropping messages.

2. Environment events. These include a *time progress* (clock tick) event and two events modeling the *attacker's capabilities*, one for modeling the attackers *collecting* and *modifying* messages and one for the attackers using these messages for *reserving bandwidth anywhere* in the network.

Below, we discuss the attacker events and one representative message processing event. For a detailed description of the remaining events, see Appendix A.5.

Attacker Events. There are two events that each attacker, i.e., malicious AS $a \in M$, can execute: (i) receive a message, partially modify it, and store the resulting message in the attacker knowledge *kwl*, and (ii) sending a message from *kwl* to any AS in the network. Recall that *kwl* also includes any message in $\mathcal{M}$ with a malicious source AS. We now discuss the two attacker events in more detail.

In the first event, CLT, the attacker $a \in M$ receives a message m from his buffer $buf(a,i)$ at interface i, possibly modifies its mutable fields *ptr* and *accBW* (but not the other fields), and adds the resulting message to *kwl*.

$$
\begin{aligned}
&CLT(m, m' \in \mathcal{M}, a \in M, i \in I) = \{(\sigma, \sigma') \mid\\
&\qquad \texttt{- guards -}\\
&\qquad m \in \sigma.buf(a,i) \wedge\\
&\qquad m' \approx m \wedge \qquad \texttt{- modify mutable fields -}\\
&\qquad \texttt{- actions -}\\
&\qquad \sigma'.kwl = \sigma.kwl \cup \{m'\}\ \}.
\end{aligned}
$$

This partial modification of the message is modeled by the relation $m \approx m'$, expressing that m and m' coincide except on their mutable fields. It precise definition is given in Appendix A.3. We assume that in an implementation of the N-Tube algorithm, the source AS signs the immutable fields with its private key.

In the second event, ATK, an attacker can send any message m in *kwl* to any neighbor AS v by adding m to v's buffers $buf(v,i)$

$$ATK(m \in \mathcal{M}, a \in M, v \in H, i, e \in I, t \in \mathbb{N}) = \{(\sigma, \sigma') \mid$$

`- guards -`

$$m \in \sigma.kwl \land$$

$$((a,e),(v,i)) \in E \land$$

`- actions -`

$$\sigma'.buf = \sigma.buf((v,i) \mapsto \sigma.buf(v,i) \cup \{m\}\ \}.$$

These two events model a powerful attacker's capabilities: Malicious ASes can attack anytime, they can make arbitrary reservation requests from their own ASes, and they can partially modify requests. Note, that these capabilities include that attackers can communicate between each other through channels outside the network to collude and synchronize attacks and they can replay old messages. However, the attackers cannot spoof arbitrary messages from honest ASes, change the reservations stored in the reservation maps of honest ASes, or change the system's global time.

Message Processing Events. We depict the CMP event, since it is the most representative message processing event. In this event, AS v receives a reservation message m at interface i (first guard) at time t (second guard), allocates bandwidth using the function *save* (first action), and forwards the modified reservation message m' using the function *send* (second action)

```
$CMP(m, m' \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N}) = \{(\sigma, \sigma') \mid$
    - guards -
(1) $m \in \sigma.buf(v, i) \wedge$
(2) $\sigma.time = t \wedge$
(3) $PathCheck(m.path) \wedge$
(4) $ResMsgCheck(m, \sigma.time) \wedge$
(5) $ResMapCheck(\sigma.res, m, v) \wedge$
(6) $m.first \leq m.ptr < m.last \wedge$
(7) $m.path[m.ptr].inI = i \wedge$
(8) $m' = process(m, \sigma.res) \wedge$
    - actions -
    $\sigma'.res = save(v, \sigma.res, m') \wedge$
    $\sigma'.buf = send(v, i, \sigma.buf, m, m')\ \}.$
```

The third, forth and fifth guard ensure that m is well-formed and compatible with existing reservations in v's reservation map that corresponds to m. Details are given in Appendix A.5. The sixth guard determines if v is on the path segment, i.e., if m's pointer is between *first* and *last*. The seventh guard checks if m traverses the path in the forward direction, i.e., if the arrival interface i of AS v matches the corresponding ingress interface given on m's path field. The final guard models, using the function *process*, how N-Tube processes the received message m resulting in m'. This function determines the available and ideal bandwidth that AS v can allocate using the functions *avail* and *ideal* described in Section 3.4, appends their results to the *accBW* field, and increments m's pointer

$$
\begin{aligned}
& process(m \in \mathcal{M}_R, resM \in ResMap) = \\
& \quad \textbf{let } newBW = (\!|\ avBW := avail(m, resM); \\
& \qquad\qquad\qquad\quad idBW := ideal(m, resM)\ |\!) \\
& \quad \textbf{in } m(\!|\ accBW := newBW \ \#\ m.accBW; ptr := m.ptr + 1\ |\!).
\end{aligned}
$$

3.6 Properties

We formally define N-Tube's five properties over valid executions and we sketch their proofs. See Appendix B for details.

Definition (Valid Executions)**.** A *valid* execution $\pi = \{(\sigma_n, \lambda_n)\}_{n\in\mathbb{N}} \in \mathfrak{E}$ satisfies the following assumptions:

Time-Progress: The global time infinitely progresses

$$\forall t \in \mathbb{N}\ \exists n \in \mathbb{N}.\ \sigma_n.time \geq t. \tag{TP}$$

Message-Progress: All messages in the buffers of honest ASes are processed in at most time *bufT*

$$\begin{aligned}&\forall n \in \mathbb{N}, v \in H, i \in I, m \in \sigma_n.buf(v,i).\\ &\quad \exists \tilde{n} > n.\ m \notin \sigma_{\tilde{n}}.buf(v,i)\ \wedge\ \sigma_{\tilde{n}}.time - \sigma_n.time \leq bufT.\end{aligned} \tag{MP}$$

This is satisfied if all honest ASes run a fair scheduling algorithm, for example Round-Robin, to prevent message starvation and messages are dropped in case of buffer overflow.

All properties assume that a *successful* reservation has been established by an honest source AS.

Definition (Successful Reservation)**.** We say *an honest source $s \in H$ makes a successful reservation confirmed by the message $m \in \mathcal{M}_R$ at time t* if the following three conditions hold:

Honest Path: m's path only contains honest ASes.

Confirmation: s confirms m at time t with sufficient bandwidth

$$\exists n \in \mathbb{N}, i \in I.\ \lambda_n = FIN(m,s,i,t) \wedge finBW(m) \geq m.minBW.$$

No deletion: There is no deletion event matching the reservation $(src(m), m.id)$ and version $m.idx$ before m's expiration time.

Theorem 23. *If an AS s makes a successful reservation m and time t, then all ASes on m's path added their* avail *and* ideal *computations to $m.accBW$ and reserve $finBW(m)$ until it expires*

$$\begin{aligned}&\forall n \in \mathbb{N}, v \in V, k \in [m.first; m.last].\\ &\quad \sigma_n.time \in\ ]t; m.expT]\ \wedge\ v = m.path[k].as\\ &\quad \Rightarrow \sigma_n.res(v,s,m.id).vrs(m.idx).resBW = finBW(m)\ \wedge\\ &\qquad \exists \tilde{n} < n, \tilde{m} \in \mathcal{M}_R.\ \tilde{m} \approx m\ \wedge\\ &\qquad\quad m.accBW[k - m.first] = (\!|\ avBW = \mathrm{avail}(\tilde{m}, \sigma_{\tilde{n}}.res);\\ &\qquad\qquad\qquad\qquad idBW = \mathrm{ideal}(\tilde{m}, \sigma_{\tilde{n}}.res)\ |\!).\end{aligned}$$

Proof Sketch. The proof proceeds by induction on n and then by case distinction on the event λ_n. By the No Deletion assumption, we do not need to consider deletion events. By the Honest Path assumption it can be shown that m only traverses honest ASes, which also excludes the ATK event. Since the reservation was successful, we can show that, for each AS v on the path from the source s back to the AS w at index $m.last$, there is exactly one message processing event, and that the ASes between $\mathit{first}(m)$ and w reserve the bandwidth $\mathit{finBW}(m)$. Likewise, going backwards from w to s, we can show that the accumulated bandwidths are as indicated. □

We model "constant bandwidth demands" using a function $\mathcal{D} : V \times \mathbb{N} \rightharpoonup_f \mathcal{M}_R$ such that $\mathcal{D}(s, id) = m$ implies $src(m) = s$ and $m.id = id$. We say that a *reservation message m corresponds to $\mathcal{D}$* if $(src(m), m.id) \in supp(\mathcal{D})$ and m coincides with $\mathcal{D}(src(m), m.id)$ on all fields except ptr, $expT$, $minBW$, and $accBW$. For the rest of this section we fix two time points $t_0, t_1 \in \mathbb{N}$ such that $t_1 - t_0 \geq 2maxT$.

Definition (Constant Demands). We say an execution $\pi \in \mathfrak{E}$ *has constant demands $\mathcal{D}$ between t_0 and t_1* if

- for all $(s, id) \in supp(\mathcal{D})$ the source AS s has successfully made a reservation confirmed by a message m corresponding to $\mathcal{D}(s, id)$ before time t_0 and successfully renews this reservation without any gaps until t_1,
- any reservation confirmed by a reservation message m between t_0 and t_1, corresponds to $\mathcal{D}$, and
- there are no deletion events between t_0 and t_1 for reservations given by $supp(\mathcal{D})$.

See Appendix A.13 for a formalization of this definition.

Theorem 24. *If there are constant demands $\mathcal{D}$ between t_0 and t_1, then after time $t_0 + 2maxT$ all reservations allocate the ideal bandwidth until t_1, i.e.,*

$$\begin{aligned}
&\exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + 2maxT\ \wedge \\
&\forall m \in rng(\mathcal{D}), r \in Res, n > \tilde{n}, v \in sgmt(m). \\
&\quad \sigma_n.time \in\ ]t_0 + 2maxT; t_1]\ \wedge\ r = \sigma_n.res_v(src(m), m.id) \\
&\qquad \Rightarrow allocBW(r, \sigma_n.time) = \min_{x \in sgmt(m)} \{\mathrm{ideal}(m, \sigma_{\tilde{n}}.res_x)\}
\end{aligned}$$

Proof sketch. We first show that after *maxT* the requested demands in all reservation maps correspond to $\mathcal{D}$. Due to the constant demands, after *2maxT* the *ideal* bandwidth allocations in the field *accBW* stay constant and are greater than the *avail* bandwidth allocations in all processed reservation messages. Hence, between t_0+*2maxT* and t_1 the *ideal* bandwidth allocation is reserved. □

3.6.1 Availability

If an honest AS makes a successful reservation m, then a positive amount of bandwidth, *finBW*(m), will be reserved on its path until it expires.

Corollary 1 (Availability)**.** Assume s makes a successful reservation m at time t, then

$$\begin{aligned}&\forall n \in \mathbb{N}, v \in \mathit{sgmt}(m).\ \sigma_n.\mathit{time} \in\ ]t; m.\mathit{expT}]\\&\Rightarrow \sigma_n.\mathit{res}_v(s, m.\mathit{id}).\mathit{vrs}(m.\mathit{idx}).\mathit{resBW} > 0.\end{aligned}$$

Proof sketch. Follows directly from Theorem 23 and the *positivity* property of the bandwidth computation from Section 3.4.3. □

3.6.2 Immutability

If an honest AS makes a successful reservation m, the reserved bandwidth stays the same for all ASes on m's path until it expires.

Corollary 2 (Immutability)**.** Assume s makes a successful reservation m at time t. Then

$$\begin{aligned}&\forall n, n' \in \mathbb{N}, v, v' \in \mathit{sgmt}(m).\ \sigma_n.\mathit{time}, \sigma_{n'}.\mathit{time} \in\ ]t; m.\mathit{expT}]\\&\Rightarrow \sigma_n.\mathit{res}_v(s, m.\mathit{id}).\mathit{vrs}(m.\mathit{idx}).\mathit{resBW}\\&\quad = \sigma_{n'}.\mathit{res}_{v'}(s, m.\mathit{id}).\mathit{vrs}(m.\mathit{idx}).\mathit{resBW}.\end{aligned}$$

Proof sketch. This follows directly from the first statement in Theorem 23, since the *resBW* fields are set to *finBW*(m) for all path ASes until m expires. □

3.6.3 Stability

During a period of constant demands, all reservations stabilize.

Corollary 3 (Stability)**.** Assume there are constant demands $\mathcal{D}$ between t_0 and t_1. Then

$$\begin{aligned}
&\forall n, n' \in \mathbb{N}, r, r' \in Res, v \in V, m \in rng(\mathcal{D}) .\\
&\ \sigma_n.time, \sigma_{n'}.time \in \,]t_0 + 2maxT; t_1] \wedge\\
&\ \ r = \sigma_n.res_v(src(m), m.id) \ \wedge \ \ r' = \sigma_{n'}.res_v(src(m), m.id)\\
&\ \ \ \Rightarrow allocBW(r, \sigma_n.time) = allocBW\left(r', \sigma_{n'}.time\right)
\end{aligned}$$

Proof sketch. This follows directly from Theorem 28. □

3.6.4 Minimum Bandwidth Guarantee

If there are constant demands $\mathcal{D}$ between t_0 and t_1, then there is a lower bound on the ideal bandwidth allocations that only depends on the request ratios on their first link and on the link capacities along their paths.

Corollary 4 (Minimum Bandwidth Guarantee)**.** Assume there are constant demands $\mathcal{D}$ between t_0 and t_1, then

$$\begin{aligned}
&\exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + 2maxT \wedge\\
&\ \forall m \in rng(\mathcal{D}), r \in Res.\\
&\ \ \exists\ G \in \,]0; 1]\, \forall n > \tilde{n}, v \in sgmt(m) \setminus \{first(m)\}.\\
&\ \ \ \sigma_n.time \in \,]t_0 + 2maxT; t_1] \ \wedge \ r = \sigma_n.res_v(src(m), m.id)\\
&\ \ \ \ \Rightarrow allocBW(r, \sigma_n.time) > G \cdot reqRatio(m, \sigma_{\tilde{n}}.res_{first(m)})
\end{aligned}$$

Proof sketch. This follows directly from Theorem 28 and the *lower* ideal *bound* property of the *ideal* function in the bandwidth computation from Section 3.4.3. □

3.6.5 Bounded Tube Fairness

If there are constant demands $\mathcal{D}$ between t_0 and t_1, then, in the absence of congestion, bandwidth of egress links is allocated proportionally between tube demands and, in case some tube demands exceed their ingress links' capacities, their tube ratio is bounded.

Corollary 5 (Bounded Tube Fairness)**.** Assume there are constant demands $\mathcal{D}$ between t_0 and t_1, then

$$\begin{aligned}
&\exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + 2maxT \wedge \\
&\quad \forall m \in rng(\mathcal{D}), v \in sgmt(m), i, i', e \in I, n > \tilde{n}. \\
&\quad tubeDem_v(i,e) \in \,]0; cap(v,i)] \wedge tubeDem_v(i',e) \in \,]0; cap(v,i')] \\
&\quad\quad \Rightarrow \frac{tubeRatio_v(i,e)}{tubeRatio_v(i',e)} = \frac{tubeDem_v(i,e)}{tubeDem_v(i',e)}.
\end{aligned}$$

Analogously, in case $tubeDem_v(i',e) \geq cap(v,i')$, e.g., there are excessive demands from i' to e,

$$\frac{tubeRatio_v(i,e)}{tubeRatio_v(i',e)} = \frac{tubeDem_v(i,e)}{cap(v,i')}.$$

Here $tubeRatio_v$ and $tubeDem_v$ denote the corresponding functions defined in Section 3.4.3 that are computed at AS v.

Proof sketch. This follows directly from Theorem 28 and the *bounded tube-proportionality* property of the *ideal* function in the bandwidth computation from Section 3.4.3. □

Bounded tube-fairness implies that, when the system reaches a stable state, the *bounded tube-proportionality* property holds globally, i.e., on all links of honest ASes. This guarantees that in case of a link-flooding allocation attack the attacked ingress links' tube ratios are always bounded, which prevents that bandwidth reservations through the other ingress links will be reduced ad infinitum.

3.7 Related work

Congestion Control enables end-to-end connections with possibly multiple paths to control their path rates to fairly mitigate congestion. The seminal work of Kelly *et al.* [45] theoretically analysed fairness and stability of the problem by framing it in terms of Network Utility Maximization (NUM). Various fairness notions, including models of deployed TCP congestion-control mechanisms, can be formulated in this setting as summarized by Srikant et al. [61]. By adding minimal information about path congestion to traversing packets, these algorithms provide resource allocation without the need for state at routers.

These works, however, are based on per-flow fairness with complete knowledge of users' utility functions. In this work we take the stance that new Internet architectures [30, 57, 70] can handle reservation states efficiently which allows them to police misbehaving traffic. As observed by Briscoe et al. [18], self-interested and strategic users can skew the overall rate allocation by opening arbitrarily many connections violating the property of minimum bandwidth guarantee. Furthermore, stateless algorithms can only reduce bandwidth allocations of misbehaving flows but cannot determine aggregated misbehaviour (over time and per AS) and cannot revoke access.

Capability-based mechanisms [33, 49, 50, 56, 69, 71] attempt to counter link-flooding attacks with authenticity identifiers, called tokens, for individual flows provided by the destination. The tokens allow routers to distinguish between legitimate and malicious DDoS traffic. However, such mechanisms are insufficient against link-flooding attacks if attackers can acquire these tokens in a legitimate manner from numerous honest and malicious destinations and send a huge number of low-volume flows through critical network links.

Game Based Mechanisms consider resource allocation for maximizing a global objective function as an "inverse game theory" problem, i.e., how to design games that achieve maximum social-welfare utility [36, 41]. Mechanism design for the resource allocation problem allows for users that are self-interested and strategic, and may attempt to manipulate the system to their advantage by misreporting information about their utility functions. Vickrey-Clarke-Groves (VCG) type mechanisms [68] provide sealed bid auctions that incentivize users to reveal their objective truthfully and can also include sellers of resources. However, for these mechanisms to allow practical bids, the class of utility functions is very restricted, e.g., to piecewise linear [36]. Furthermore, the main objective of these mechanisms is to increase efficiency and not to provide minimal guarantees.

Resource reservation architectures such as RSVP [72], DiffServ [35], and PCE [67] enable users to make bandwidth reservations along network paths by allowing reservation state at the routers. However, they neither handle malicious reservations nor do their authors provide formal arguments to support their claims.

For networks supporting path-based forwarding, SIBRA [10], the approach closest to our work, describes a scalable inter-domain bandwidth allocation architecture that allows ASes to make and enforce reservations along network paths. The SIBRA architecture is based on a *distributed bandwidth reservation algorithm* that processes reservation requests made by ASes and allocates bandwidth on the path's links accordingly. Together with an *enforcement mechanism* that monitors and polices these reservations, the authors claim that this architecture provides an effective QoS scheme, i.e., that it provides *minimum bandwidth guarantees*, which they call *botnet-size independence*. However, their reservation algorithm is based on proportional fair bandwidth sharing, allowing attackers to reserve excessive amounts of bandwidth and to reduce legitimate reservation arbitrarily. Without formal arguments their claim to provide *botnet-size independence* remains questionable.

3.8 Conclusion and future work

We have presented N-Tube, the first provably correct, distributed bandwidth reservation algorithm. N-Tube makes a substantial step forward to providing a reliable counter-measure against DDoS attacks in the presence of powerful attackers. By rigorously verifying safety and security properties, our algorithm rests on a solid foundation where all corner cases are considered and checked, which dramatically reduces the risk of future vulnerabilities and exploits. Another key contribution is the new fairness notion of bounded tube-fairness that provides minimal bandwidth guarantees in the face of excessive demands and proportional fairness under normal circumstances with moderate demands.

As future work we would like to refine the underlying stability assumptions, to couple our fairness notion with the Network Utility Maximization setting, and to verify all proofs using a theorem prover. We are also working on integrating the N-Tube algorithm into an existing path-based forwarding architecture to finally enter the era of DDoS free Internet.

Chapter 4
Conclusion

4.1 Summary

As shown in Verizon's data breach report from 2019 [2], large organizations and users of today's Internet most frequently suffer from severe security incidents due to privilege misuse and denial of service. To address these two problems, sophisticated techniques were invented in the field of *policy analysis* to provide **qualitative guarantees** for access control policies of large organizations and in the field of *bandwidth reservation* to provide **quantitative guarantees** for Internet users during DDoS attacks. However, policy analysis suffers from an imbalance between expressiveness of policy specification and the efficiency of policy analysis. Furthermore, bandwidth reservation schemes neither handle malicious reservations nor do they provide formal arguments to support their claimed guarantees. In this thesis, we proposed two solutions, one for each challenge.

First, we proposed FORBAC, a framework that strikes a balance between expressiveness and efficient policy analysis for RBAC. FORBAC is expressive enough to formalize a wide range of access control policies. However, it is also simple enough so that relevant policy analysis queries can be analyzed in NP, which we argue is a natural complexity class for this problem. To analyze queries efficiently, we reduce them to the problem of satisfiability modulo appropriate theories, and use off-the-shelf SMT solvers. FORBAC is the first framework that provides an expressive access control policy language together with a reasonable complexity class for its policy analysis. Furthermore, we provide strong evidence that our approach is practical by evaluating it in a case study with a large European bank.

Second, we proposed N-Tube, the first provably correct, distributed bandwidth reservation algorithm. We formalized and proved that N-Tube's bandwidth allocations *quickly stabilize* in periods of constant demands and that the resulting stable state (i) guarantees the allocation of a *minimum bandwidth*, even in the presence of congestion, and (ii) satisfies a new fairness notion called *bounded tube-fairness*. We also proved that any successful reservation immediately reserves some bandwidth and existing reservations are *immutable* up to their expiration time. By rigorously verifying safety and security properties, our algorithm rests on a solid foundation where all corner cases are considered and checked, which dramatically reduces the risk of future vulnerabilities and exploits.

4.2 Future Work

Working on these solutions has led to the following open challenges for future research directions.

Extending the FORBAC-language with further access control idioms: As future work, we propose to extend FORBAC with two RBAC idioms that have received attention in the literature. The first is role hierarchies, which define a partial order on a set of roles. Roles are assigned permissions granted to that role and in addition, permissions granted to all its subroles in the hierarchy. There is no standard that specifies how to define role hierarchies with role templates. The second idiom consists of two types of constraints: static and dynamic separation-of-duty constraints and cardinality constraints. Further idioms which are provided by the XACML language are delegation and policy composition. Delegation allows to transfer access rights among users. Policy composition provides multi-valued decisions and rules how to resolve conflicts between contradicting decisions of sub-policies. It remains open how to define these idioms in RBAC models where users, roles, and permissions have attributes.

Providing complexity bounds for policy analysis of XACML policies: In the work of Athena+Yices [7] the authors formalize XACML policies like the CONTINUE [47] policy with first-order logic. They also provide new policy properties, like observational equivalence, conflict detection, and change-impact analysis. However, the authors do not provide complexity bounds for their policy analysis problems which are undecidable in general. It remains open how to define extensions of the FORBAC fragment that can specify XACML policies together with these properties and what the complexity bounds are for their policy analysis.

Formally verify the N-Tube algorithm in a theorem prover: We provided a principled solution to the resource reservation problem: First we modelled the N-Tube algorithm and a powerful attacker as a labeled transition system. Second, we formalized and proved that N-Tube's bandwidth allocations provides quantitative guarantees, i.e., stability and minimum bandwidth allocations. The third step remains for future research to formally verify all these proofs using a theorem prover. This guarantees that all corner-cases were considered and verified correctly. In addition, we are working on integrating the N-Tube algorithm into an existing path-based forwarding architecture, namely SCION. Another research direction is to combine program verification of its implementation with our formalization.

Framing the N-Tube bandwidth allocation computation in the Network Utility Maximization setting: Our newly proposed notion of *bounded tube-fairness* provides a local property, i.e., it holds for allocations at each AS on the path of a reservation request. Our approach of introducing N-Tube is similar to the research development of congestion control in the early Internet. Due to the problem of *congestion collapse,* Van Jacobsen started the development of congestion control for TCP in the mid 1980s [61]. But much later in the mid 1990s a formal framework was designed by Kelly *et al.* to provide a mathematical model by framing it in terms of Network Utility Maximization (NUM) [45]. Later works extended this approach using game-based mechanisms which consider resource allocation as an "inverse game theory" problem, i.e., how to design games that achieve maximum social-welfare utility [36, 41]. It remains to show if *bounded tube-fairness* can also be expressed in the setting of an optimization problem maximizing a global objective function, where N-Tube approximates its maximum in a distributed manner.

Appendix A
Appendix: Model

A.1 Network and Environment

The network is modeled as a *directed, labeled multi-graph*. We provide a more refined model using *arcs A* and two corresponding functions *src* and *hd* to define a network:

Definition 25. *Given three finite sets V, A, and I. We define a* network $\eta \in \mathcal{N}$ *as a record with*

$$\mathcal{N} = (\!|\ ases = V; links = A; intf = I;\\ tl, hd \in A \to V \times I; cap \in A \to \mathbb{R}_0^+\ |\!)$$

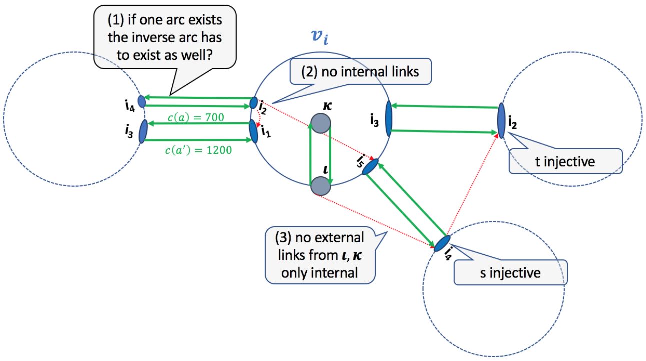
with the following components:

- *V is a finite set of vertices, called ASes.*
- *The nodes of the graph are given by the set $V \times I$.*
- *Hereby, the finite set I provides a global set of identifiers, which are used for interfaces inside of each AS.*
- *The finite set of arcs A is called* links *being the domain for the following functions:*
  - *The weight of each link is given by the function $cap : A \to \mathbb{R}_0^+$, the* capacity function.
  - *A link can start from exactly one interface of an AS given by the injective function $tl : A \to V \times I$ and*
  - *ends in at exactly one interface of another AS given by the injective function $hd : A \to V \times I$.*
- *To guarantee that G is a valid network graph the following constraints must hold:*
  - *No internal links*

$$\forall u, v \in V, i, j \in I, a \in A.\ tl(a) = (u, i) \wedge hd(a) = (v, j) \Rightarrow u \neq v$$

– *Inverse links*

$$\begin{aligned}&\forall u,v \in V, i,j \in I, a \in A.\\ &\quad tl(a) = (u,i) \wedge hd(a) = (v,j)\\ &\qquad \Rightarrow \exists a' \in A.\ tl(a') = (v,j) \wedge hd(a') = (u,i)\end{aligned}$$



Definition 26. *The set* environment Γ *is defined as*

$$\Gamma = \llparenthesis\ \eta \in \mathcal{N}; \delta \in]0;1[; maxT, bufT \in \mathbb{R}_0^+\ \rrparenthesis.$$

An environment $\gamma \in \Gamma$ contains the network graph η, the N-Tube parameters δ, to adjust link capacities, maxT, limiting the maximum time a version of a reservation can be valid, and bufT, limiting the maximum time a message stays in a buffer until it is processed.

A.2 Paths

For any $p \in \mathcal{P}$ it has to hold that it

- contains at at least one link, i.e., $length(p) \geq 1$
- is directed and consistent with the network

$$\begin{aligned}&\forall k \in \mathbb{N}, u,v \in V, i,e,i',e' \in I.\\ &\quad 0 \leq k \leq length(p) - 1 \wedge\\ &\quad p[k] = u_i^e \ \wedge\ p[k+1] = v_{i'}^{e'}\\ &\qquad \Rightarrow\ \exists a \in A.\ src(a) = (u,e)\ \wedge\ hd(a) = (v,i')\end{aligned}$$

- is loop-free

$$\forall k, k' \in \mathbb{N}, u, v \in V, i, e, i', e' \in I.$$
$$k < k' \leq \mathit{length}(p) \ \wedge \ p[k] = u_i^e \ \wedge \ p[k'] = v_{i'}^{e'} \ \Rightarrow \ u \neq v$$

The set of ASes on a path p are defined by

$$\mathit{nodes}(p) := \{v \in V \mid \exists k \in \mathbb{N}.\ \ p[k].\mathit{as} = v\}$$

In this work we only consider the set of valid paths $\mathcal{P}$ and ignore the underlying network η.

A.3 Messages

As described before *messages* $\mathcal{M}$ are defined as either *deletion messages* or *reservation messages*

$$\mathcal{M} = \ \mathcal{M}_D + \mathcal{M}_R$$

together with injections $\mathit{delM} : \mathcal{M}_D \to \mathcal{M}$ and $\mathit{resM} : \mathcal{M}_R \to \mathcal{M}$.
For a valid message m the following must hold:

1. Valid path, i.e., $m.\mathit{path} \in \mathcal{P}$
2. Valid path counter
$$m.\mathit{ptr} \leq \mathit{length}(m.\mathit{path})$$
3. Valid pointers
$$m.\mathit{first} < m.\mathit{last} \leq \mathit{length}(m.\mathit{path})$$
4. Valid current bandwidth
$$\mathit{length}(m.\mathit{accBW}) = \max(0, m.\mathit{ptr} - m.\mathit{first})$$
5. Valid bandwidth range
$$m.\mathit{minBW} \leq m.\mathit{maxBW} \ \wedge \ 0 < m.\mathit{maxBW}$$

The function $src : \mathcal{M} \to V$ extracts the source AS from a message's path.

$$src(m) = m.path[0].as$$

The function $cur : \mathcal{M} \to \llparenthesis\ inI \in I; as \in V; egI \in I\ \rrparenthesis$ extracts the current AS together with the corresponding ingress and egress interface from a message m.

$$cur(m) = m.path[m.ptr]$$

The function $nodes : \mathcal{M} \to \mathcal{P}(V)$ extracts the AS between on the *path* field of a message,

$$nodes(m) = \{v \in V \mid \exists k \in [0; length(m.path)).\ v = m.path[k].as\}.$$

The function $sgmt : \mathcal{M}_R \to \mathcal{P}(V)$ extracts the AS between *first* and *last* of a reservation message,

$$sgmt(m) = \{v \in V \mid \exists k \in [m.first; m.last].\ v = m.path[k].as\}.$$

Given two messages $m, m' \in \mathcal{M}$ We say *m corresponds to m′* (and reversely) if they refer to the same version of a reservation

$$\begin{aligned} &m \sim m' :\Longleftrightarrow \\ &\quad src(m) = src(m') \ \wedge\ m.id = m'.id \ \wedge\ m.idx = m'.idx \end{aligned}$$

Given two reservation messages $m, m' \in \mathcal{M}_R$. We say *m is equivalent to m′* (and reversely) if all fields except of their *ptr* and *accBW* coincide

$$\begin{aligned} &m \approx m' \ :\Longleftrightarrow \\ &\quad m.id = m'.id \ \wedge \\ &\quad m.idx = m'.idx \ \wedge \\ &\quad m.path = m'.path \ \wedge \\ &\quad m.first = m'.first \ \wedge \\ &\quad m.last = m'.last \ \wedge \\ &\quad m.minBW = m'.minBW \ \wedge \\ &\quad m.maxBW = m'.maxBW \ \wedge \\ &\quad m.expT = m'.expT \end{aligned}$$

Note that the relations $\approx \subseteq \mathcal{M}_R \times \mathcal{M}_R$ and $\sim \subseteq \mathcal{M} \times \mathcal{M}$ are equivalence relations.

A.4 Reservation Maps

A valid reservation map has to be consistent regarding their reservations in the following sense:

$$\begin{aligned}
&\forall v,s \in V, id, k, st, en \in \mathbb{N}, p \in \mathcal{P}, vrs \in VrsMap\\
&\quad \sigma.res(v,s,id) = (\!|\, p, k, st, en, vrs \,|\!) \land\\
&\quad p[k].as = v \ \land \ p \in \mathcal{P} \ \land \ s = p[0].as \ \land \ st \le k \le en \le length(p) \land\\
&\quad \forall idx \in \mathbb{N}, \min, \max, idl, res \in \mathbb{R}_0^+, expT \in \mathbb{N}.\\
&\quad\ \ vrs(idx) = (\!|\, \min, \max, idl, res, expT \,|\!)\\
&\quad\ \ \ \min \le res \le \max \ \land 0 < \max
\end{aligned}$$

and it must hold that the reserved bandwidth for any egress link does not exceed the link's capacity, i.e.,

$$\begin{aligned}
&\forall v \in V, e \in I, t \in \mathbb{N}.\\
&\textstyle\sum_{\substack{r \in rng(\sigma.res_v):\\ resEg(r)=e}} allocBW(r.vrs, t) \le cap(v, e).
\end{aligned}$$

The functions $resSr : Res \to V$, $resEg : Res \to I$ and $resIn : Res \to I$ extract the corresponding reservation's source AS and egress and ingress interface.

$$\begin{aligned}
resSr(r) &= r.path[0].as\\
resEg(r) &= r.path[r.ptr].egI\\
resIn(r) &= r.path[r.ptr].inI
\end{aligned}$$

The function $sgmt : Res \to \mathcal{P}(V)$ ASes on the *path segment of a reservation* are given by the following function:

$$sgmt(r) = \{v \in V \mid \exists k \in [r.first; r.last].\ v = r.path[k].as\}.$$

Given a reservation $r \in Res$ and a reservation message $m \in \mathcal{M}_R$. We say r *corresponds to m* if holds

$$\begin{aligned}
&r \propto m \ :\Leftrightarrow\\
&\quad m.maxBW = r.maxBW \land\\
&\quad m.path = r.path \land\\
&\quad m.first = r.first \land\\
&\quad m.last = r.last
\end{aligned}$$

A.5 Events

There are 14 events in our model. In contrast to the rest of the events, the first event *TCK* and the second event *RST* are not triggered by a message arrival, but model time progress and expiration of reservations, respectively.

In case the arrived message is a reservation message the following six events can be triggered.

The event *FWD* describes how the message is forwarded along the path until it reaches the AS where the N-Tube algorithm starts reserving bandwidth. The event *CMP* continues from there and describes how the bandwidth allocation is computed by N-Tube, how the results are added to the message and how the updated message is then forwarded until the end. The event *TRN* finishes the N-Tube computation at the last AS of the reservation and sends the updated message backwards along the path.

The event *UPT* describes how the reservations are updated in the reservation maps of each AS between the end and including the start of path on the trip back. The event *BWD* sends the message backward along the path and the event *FIN* finally drops the message at the source.

In case the arrived message is a deletion message two events are triggered. In the event *RMV* all ASes forward the deletion message along the path and delete the corresponding version of the reservation from their reservation maps. The event *DST* is triggered at the destination of the path and instead of forwarding the message it drops it.

At any arrival of an reservation or deletion messages the event *DRP* can be triggered that just drops the message without any interaction with the N-Tube algorithm. Furthermore, we define two attack events *ATK* and *CLT* as described previously.

Tick event: This is the only event that models the progress of time in the system by increasing the state's *time*-field to progress to its successor state.

$$
\begin{aligned}
&TCK(t \in \mathbb{N}) = \{(\sigma, \sigma') \mid \\
&\qquad \texttt{- guards -} \\
&\qquad \sigma.time = t \ \wedge \\
&\qquad \texttt{- actions -} \\
&\qquad \sigma'.time = \sigma.time + 1 \ \}
\end{aligned}
$$

Reset event: This event models the removal of expired or unsuccessful reservations. Given by the event's parameters a reservation at an honest AS v and identified by source s, id, and idx, this event can trigger non-deterministically for the corresponding reservation. Its first guard is satisfied if the reservation has been expired before the current time of the system. Its second is satisfied is satisfied if less bandwidth was reserved than the source AS asked for.

```
$RST(v \in H, s \in V, id, idx \in \mathbb{N}) = \{(\sigma, \sigma') \mid$
    - guards -
    $( \sigma.res(v,s,id).vrs(idx).expT < \sigma.time \lor$
      $\sigma.res(v,s,id).vrs(idx).resBW$
          $< \sigma.res(v,s,id).vrs(idx).minBW ) \land$
    - actions -
    $\sigma'.res = delRes(v,s,id,idx,\sigma.res) \}$
```

The function *delRes* removes the version *idx* of the reservation identified by (s, id) from v's reservation map:

```
$delRes(v, s \in V,\ id, idx \in \mathbb{N}, res \in ResMap) =$
    **let**
     $delVrs := res(v,s,id).vrs\,(idx \mapsto \bot)$
    **in**
     $res\big((v,s,id) \mapsto (\!|\ vrs := delVrs\ |\!)\big)$
```

A.6 Event Guards

For message creation and message processing events a set of checks are executed, given in as event guards. In the following the predicates for these guards are presented:

PathCheck : $\mathcal{P} \to \mathbb{B}$ checks the validity of the path *m.path*, i.e., if it contains at at least one link, it is *loop-free*, and it is *consistent* with the network (all its links have positive capacities)

$$\begin{aligned}
&\mathit{PathCheck}(p \in \mathcal{P}) = \\
&\ \mathit{length}(p) \geq 1 \ \land \\
&\ \forall k \in \mathbb{N}, u \in V, i, e, i', e' \in I. \\
&\quad k < \mathit{length}(p) \ \land \ p[k] = u_i^e \ \land \ p[k+1] = {u'}_{i'}^{e'} \\
&\quad \Rightarrow \mathit{cap}(u,i) > 0 \ \land \ \mathit{cap}(u,e) > 0 \ \land \\
&\ \forall k, k' \in \mathbb{N}, u, u' \in V, i, e, i', e' \in I. \\
&\quad k < k' \leq \mathit{length}(p) \ \land \ p[k] = u_i^e \ \land \ p[k'] = {u'}_{i'}^{e'} \\
&\quad \Rightarrow \ u \neq u'
\end{aligned}$$

Loc : $\mathcal{M} \to \mathbb{B}$ checks at which part the of the path *m.path* the AS *v* arrived. There are five instantiations of this predicate:

$$\begin{aligned}
&\mathit{atSrc}(m \in \mathcal{M}) = \\
&\qquad m.\mathit{ptr} = 0
\end{aligned}$$

$$\begin{aligned}
&\mathit{onWay}(m \in \mathcal{M}) = \\
&\qquad 0 < m.\mathit{ptr} < m.\mathit{first}
\end{aligned}$$

$$\begin{aligned}
&\mathit{atSrt}(m \in \mathcal{M}) = \\
&\qquad m.\mathit{ptr} = m.\mathit{first}
\end{aligned}$$

$$\begin{aligned}
&\mathit{onPth}(m \in \mathcal{M}) = \\
&\qquad m.\mathit{first} < m.\mathit{ptr} < m.\mathit{last}
\end{aligned}$$

$$\begin{aligned}
&\mathit{atEnd}(m \in \mathcal{M}) = \\
&\qquad m.\mathit{ptr} = m.\mathit{last}
\end{aligned}$$

Note that by assumption (D) all these are disjoint predicates, i.e. non of their conjunctions is satisfiable.

Dir : $V \times I \times \mathcal{M}_R \to \mathbb{B}$ checks at which border router the message m arrived compared to the provided path *m.path* and if the length of the *accBW* field fits the position of v on the path:

$$\begin{aligned}
&\mathit{isFwd}(v \in V, i \in I, m \in \mathcal{M}_R) = \\
&\qquad m.\mathit{path}[m.\mathit{ptr}].\mathit{as} = v \ \wedge \\
&\qquad m.\mathit{path}[m.\mathit{ptr}].\mathit{inI} = i \ \wedge \\
&\qquad \mathit{length}(m.\mathit{accBW}) = \max(0, m.\mathit{ptr} - m.\mathit{first})
\end{aligned}$$

$$\begin{aligned}
&\mathit{isBwd}(v \in V, i \in I, m \in \mathcal{M}_R) = \\
&\qquad m.\mathit{path}[m.\mathit{ptr}].\mathit{as} = v \ \wedge \\
&\qquad m.\mathit{path}[m.\mathit{ptr}].\mathit{egI} = i \ \wedge \\
&\qquad \mathit{length}(m.\mathit{accBW}) = m.\mathit{last} - m.\mathit{first}
\end{aligned}$$

$$\begin{aligned}
&\mathit{isWrg}(v \in V, i \in I, m \in \mathcal{M}_R) = \\
&\qquad \neg(\mathit{isFwd}(v, i, m) \vee \mathit{isBwd}(v, i, m))
\end{aligned}$$

$\mathit{Rsvd} : \mathit{ResMap} \times V \times \mathcal{M}_R \times \mathbb{N} \rightarrow \mathbb{B}$ checks if there is already a valid version of the reservation corresponding to the arrived reservation message in the reservation map. The predicate *isRsvd* defines a valid version of a reservation r, i.e., successful and not expired.

$$\begin{aligned}
&\mathit{isRsvd}(res, v, m, t) = \\
&\quad \exists r \in \mathit{Res}.\ r = res(v, \mathit{src}(m), m.\mathit{id}) \ \wedge \\
&\qquad \neg(0)\ r \neq \bot \ \wedge \\
&\qquad \neg(5)\ r.\mathit{vrs} \neq \emptyset \ \wedge \\
&\qquad \neg(6)\ r.\mathit{vrs}(m.\mathit{idx}) \neq \bot \ \wedge \\
&\qquad (9)\ r.\mathit{vrs}(m.\mathit{idx}).\mathit{expT} \geq t \ \wedge \\
&\qquad (10)\ r.\mathit{vrs}(m.\mathit{idx}).\mathit{resBW} \geq r.\mathit{vrs}(m.\mathit{idx}).\mathit{minBW} \ \wedge \\
&\qquad (11)\ r.\mathit{vrs}(m.\mathit{idx}).\mathit{resBW} \geq \min(m.\mathit{accBW})
\end{aligned}$$

The predicate *isRsvd* defines a marked version of a reservation r to indicate that a version of the reservation with the corresponding index $m.idx$ has been made before.

$$\begin{aligned}
&\mathit{isMrkd}(res, v, m, t) = \\
&\quad \exists r \in \mathit{Res}.\ r = res(v, src(m), m.id) \wedge \\
&\quad\quad \neg(0)\ r \neq \bot \wedge \\
&\quad\quad \neg(5)\ r.vrs \neq \emptyset \wedge \\
&\quad\quad \neg(6)\ r.vrs(m.idx) \neq \bot \wedge \\
&\quad\quad (9)\ r.vrs(m.idx).expT \geq t \wedge \\
&\quad\quad (10')\ r.vrs(m.idx).resBW = \bot
\end{aligned}$$

The predicate *notRsvd* indicates that no version of the reservation r exists yet or that one has existed but was unsuccessful or has been expired.

$$\begin{aligned}
&\mathit{notRsvd}(res, v, m, t) = \\
&\quad \exists r \in \mathit{Res}.\ r = res(v, src(m), m.id) \wedge \\
&\quad (\ (0)\ r = \bot \vee \\
&\quad\quad (5)\ r.vrs = \emptyset \vee \\
&\quad\quad (6)\ r.vrs(m.idx) = \bot \vee \\
&\quad\quad \neg(9)\ r.vrs(m.idx).expT < t \vee \\
&\quad\quad \neg(10)\ r.vrs(m.idx).resBW < r.vrs(m.idx).minBW \vee \\
&\quad\quad \neg(11)\ r.vrs(m.idx).resBW < \min(m.accBW)\ )
\end{aligned}$$

It holds that all three event are mutual exclusive and cover all cases:

Lemma 1.

$$\mathit{isMrkd} \wedge \mathit{isRsvd} \Longleftrightarrow \text{FALSE}$$
$$\neg(\mathit{isMrkd} \vee \mathit{isRsvd}) \Longleftrightarrow \mathit{notRsvd}$$

ResMsgCheck : $\mathcal{M}_R \times \mathbb{N} \times \mathbb{N} \to \mathbb{B}$ is checked only if m is a reservation message as follows:

$$\begin{aligned}
&\mathit{ResMsgCheck}(m \in \mathcal{M}_R, t, maxT \in \mathbb{N}) = \\
&\quad m.expT - maxT \leq t < m.expT \wedge \\
&\quad m.minBW \leq m.maxBW \wedge \\
&\quad 0 < m.maxBW \wedge \\
&\quad m.first < m.last \leq length(m.path)
\end{aligned}$$

ResMapCheck : *ResMap* $\times V \times \mathcal{M}_R \rightarrow \mathbb{B}$ compares the $path$, and the pointers ptr, *first* and *last* of a message m with the corresponding reservation in res of v at the entry $m.id$ as follows:

$$
\begin{aligned}
& \mathit{ResMapCheck}(res, v, m) = \\
& \quad \exists r \in \mathit{Res}.\ r = res(v, src(m), m.id) \land \\
& \qquad (0)\ r = \bot \lor \\
& \qquad (\neg(0)\ r \neq \bot \land \\
& \qquad\ (1)\ r.path = m.path \land \\
& \qquad\ (2)\ r.ptr = m.ptr \land \\
& \qquad\ (3)\ r.first = m.first \land \\
& \qquad\ (4)\ r.last = m.last \land \\
& \qquad\ ((5)\ r.vrs = \emptyset \lor \\
& \qquad\ \ \neg(5)\ r.vrs \neq \emptyset \land \\
& \qquad\ \ \neg(6)\ r.vrs(m.idx) \neq \bot \land \\
& \qquad\ \ (7)\ r.vrs(m.idx).minBW = m.minBW \land \\
& \qquad\ \ (8)\ r.vrs(m.idx).minBW = m.maxBW \land \\
& \qquad\ \ (9)\ r.vrs(m.idx).expT = m.expT\ )\ )
\end{aligned}
$$

After the message processing events on the back traversal of the path it holds that (1) the reservation message was valid and (2) there has been a corresponding reservation, i.e.,

$$\mathit{ResMapCheck}(res, v, m) \land \mathit{isRsvd}(res, v, m, t)$$

This can be summarized in the property:

$$
\begin{aligned}
&\mathit{isStrongRsvd}(res, v, m, t) = \\
&\qquad r = res(v, src(m), m.id) \wedge \\
&\qquad \neg(0)\ r \neq \bot \wedge \\
&\qquad (1)\ r.path = m.path \wedge \\
&\qquad (2)\ r.ptr = m.ptr \wedge \\
&\qquad (3)\ r.first = m.first \wedge \\
&\qquad (4)\ r.last = m.last \wedge \\
&\qquad \neg(5)\ r.vrs \neq \emptyset \\
&\qquad \neg(6)\ r.vrs(m.idx) \neq \bot \wedge \\
&\qquad (7)\ r.vrs(m.idx).minBW = m.minBW \wedge \\
&\qquad (8)\ r.vrs(m.idx).maxBW = m.maxBW \wedge \\
&\qquad (9)\ r.vrs(m.idx).expT \geq t
\end{aligned}
$$

The following Lemma shows that the predicate *isStrongRsvd* is equivalent to $\mathit{ResMapCheck}(res, v, m)$ together with $\mathit{isRsvd}(res, v, m, t)$:

Lemma 2.

$$
\begin{aligned}
&\mathit{ResMapCheck}(res, v, m) \wedge \mathit{isRsvd}(res, v, m, t) \\
&\qquad \Leftrightarrow \mathit{isStrongRsvd}(res, v, m, t)
\end{aligned}
$$

$\mathit{ResMapCheck}_D : \mathit{ResMap} \times V \times \mathcal{M}_D \rightarrow \mathbb{B}$ is the analogous predicate for deletion messages which only keeps checks $(1-4)$, since for a deletion message m does not contain the fields *minBW*, *maxBW*, and *expT*.

$$
\begin{aligned}
&\mathit{ResMapCheck}(res, v, m) = \\
&\qquad \exists r \in Res.\ r = res(v, src(m), m.id) \wedge \\
&\qquad\quad (0)\ r = \bot \vee \\
&\qquad\quad (\neg(0)\ r \neq \bot \wedge \\
&\qquad\quad\ (1)\ r.path = m.path \wedge \\
&\qquad\quad\ (2)\ r.ptr = m.ptr \wedge \\
&\qquad\quad\ (3)\ r.first = m.first \wedge \\
&\qquad\quad\ (4)\ r.last = m.last\ )
\end{aligned}
$$

A.7 Message-Creation

The two *message creation* events describe how honest ASes create reservation and deletion messages correctly. The creation of reservation messages is defined as follows:

```
$CRT_R(\gamma \in \Gamma, m \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N})$
 $= \{(\sigma, \sigma') \mid$
     - guards -
     $\sigma.time = t \land$
     $PathCheck(m.path) \land$
     $ResMsgCheck(m, \sigma.time, \gamma.maxT) \land$
     $ResMapCheck(\sigma.res, m, v) \land$
     $atSrc(resMsg(m)) \land$
     $isFwd(v, i, m) \land$
     $\neg isMrkd(\sigma.res, v, resMsg(m), \sigma.time) \land$
     - actions -
     $\sigma'.buf = \sigma.buf\,((v, i) \mapsto \sigma.buf(v, i) \cup \{m\}) \land$
     $\sigma'.res = mark(v, \sigma.res, m)\ \}$
```

AS v creates a *valid* reservation message m at time t and adds it to the buffer of the ingress interface i. To avoid that it creates a further reservation messages with the same identifiers $src(m)$, $m.id$, and $m.idx$, the source marks the corresponding version of the reservation in its reservation map when creating m. The guard $\neg isMrkd$ together with the function *mark* guarantee that no duplicates are created as long as m is not expired.

$$
\begin{aligned}
&\mathit{mark}(v \in V, \mathit{resM} \in \mathit{ResMap}, m \in \mathcal{M}_R) = \\
&\quad \textbf{let} \\
&\quad\; \mathit{newVrs} = \llparenthesis\, \mathit{minBW} := m'.\mathit{minBW}; \\
&\qquad\qquad\qquad \mathit{maxBW} := m'.\mathit{maxBW}; \\
&\qquad\qquad\qquad \mathit{idBW} := \bot; \\
&\qquad\qquad\qquad \mathit{resBW} := \bot; \\
&\qquad\qquad\qquad \mathit{expT} := m'.\mathit{expT} \,\rrparenthesis \\
&\quad\; \mathit{newVrsM} = \mathit{resM}(v, \mathit{src}(m), m'.\mathit{id}).\mathit{vrs}\big(m'.\mathit{idx} \mapsto \mathit{newVrs}\big) \\
&\quad\; \mathit{newRes} = \llparenthesis\, \mathit{path} := m'.\mathit{path}; \\
&\qquad\qquad\qquad \mathit{ptr} := m'.\mathit{ptr}; \\
&\qquad\qquad\qquad \mathit{first} := m'.\mathit{first}; \\
&\qquad\qquad\qquad \mathit{last} := m'.\mathit{last}; \\
&\qquad\qquad\qquad \mathit{vrs} := \mathit{newVrsM} \,\rrparenthesis \\
&\quad \textbf{in} \\
&\quad\; \mathit{resM}\big((v, \mathit{src}(m), m'.\mathit{id}) \mapsto \mathit{newRes}\big)
\end{aligned}
$$

The creation of deletion messages is similarly defined, but less guards are sufficient:

$$
\begin{aligned}
&\mathit{CRT}_D(\gamma \in \Gamma, m \in \mathcal{M}_D, v \in H, i \in I, t \in \mathbb{N}) \\
&\; = \{(\sigma, \sigma') \mid \\
&\qquad \texttt{- guards -} \\
&\qquad \sigma.\mathit{time} = t \;\wedge \\
&\qquad \mathit{PathCheck}(m.\mathit{path}) \;\wedge \\
&\qquad \mathit{atSrc}(\mathit{delMsg}(m)) \;\wedge \\
&\qquad \mathit{isFwd}(v, i, \mathit{delMsg}(m)) \;\wedge \\
&\qquad \texttt{- actions -} \\
&\qquad \sigma'.\mathit{buf} = \sigma.\mathit{buf}\,((v,i) \mapsto \sigma.\mathit{buf}(v,i) \cup \{m\})\; \} \;\wedge \\
&\qquad \sigma'.\mathit{res} = \mathit{mark}(v, \sigma.\mathit{res}, m)\; \}
\end{aligned}
$$

Note that we do check if there is a corresponding reservation in v's reservation map.

A.8 Reservation Message Arrival Events

These events are triggered by a reservation message arrival and are given by the following event template *RES*:

$$
\begin{aligned}
&\mathit{RES}(\gamma \in \Gamma, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N},\\
&\qquad \mathit{Loc} \in \mathcal{M} \to \mathbb{B},\\
&\qquad \mathit{Dir} \in V \times I \times \mathcal{M}_R \to \mathbb{B},\\
&\qquad \mathit{Rsvd} \in \mathit{ResMap} \times V \times \mathcal{M}_R \times \mathbb{N} \to \mathbb{B},\\
&\qquad \mathit{updMsg} \in \mathcal{M}_R \times \mathit{ResMap} \times]0;1[\times \mathbb{N} \to \mathcal{M}_R,\\
&\qquad \mathit{updBuf} \in V \times I \times \mathit{Buff} \times \mathcal{M} \times \mathcal{M} \to \mathit{Buff},\\
&\qquad \mathit{updRes} \in V \times \mathit{ResMap} \times \mathcal{M}_R \to \mathit{ResMap})\\
&= \{(\sigma, \sigma') \mid\\
&\qquad \texttt{- guards -}\\
&\qquad m \in \sigma.\mathit{buf}(v, i) \wedge\\
&\qquad \sigma.\mathit{time} = t \wedge\\
&\qquad \mathit{PathCheck}(m.\mathit{path}) \wedge\\
&\qquad \mathit{ResMsgCheck}(m, \sigma.\mathit{time}, \gamma.\mathit{maxT}) \wedge\\
&\qquad \mathit{ResMapCheck}(\sigma.\mathit{res}, m, v) \wedge\\
&\qquad \mathit{Loc}(\mathit{resMsg}(m)) \wedge\\
&\qquad \mathit{Dir}(v, i, m) \wedge\\
&\qquad \mathit{Rsvd}(\sigma.\mathit{res}, v, m, \sigma.\mathit{time}) \wedge\\
&\qquad m' = \mathit{updMsg}(m, \sigma.\mathit{res}, \gamma.\delta, \sigma.\mathit{time}) \wedge\\
&\qquad \texttt{- actions -}\\
&\qquad \sigma'.\mathit{buf} = \mathit{updBuf}(v, i, \sigma.\mathit{buf}, \mathit{resMsg}(m), \mathit{resMsg}(m')) \wedge\\
&\qquad \sigma'.\mathit{res} = \mathit{updRes}(v, \sigma.\mathit{res}, m') \;\}
\end{aligned}
$$

This event-template should be understood as follows:

- A reservation message m arrives at AS v at interface i at time $\sigma.\mathit{time}$ to be processed, i.e., $m \in \sigma.\mathit{buf}(v, i)$ and $\sigma.\mathit{time} = t$.

- The predicates *PathCheck* and *ResMsgCheck* ensure that the path $m.\mathit{path}$ and the m are well-formed. The predicate *ResMapCheck* ensures that the arriving reservation message m fits an already existing reservation with ID $m.\mathit{id}$ in v's reservation map.

- The function *updMsg* given as a parameter creates a new reservation message m' from the arrived message m. Here the N-Tube computation is executed and the pointer *m.ptr* is updated depending on the location and the direction of the message on the path.

- The predicates *Loc*, *Dir*, and *Rsvd* indicate at which part of the path *m.path* the message is arrived, at which interface it arrived and if there is already an existing reservation for *m.id* and *m.idx* in v's reservation map.

- Using the function *updBuf* the processed message m' is either sent to the next AS on the path given by *m.path*. or is dropped after it returned to the source.

- The function *updRes* saves the reservation given by m' in v's reservation map between *m.first* and *m.last* on the path, and marks it otherwise.

Forward event: This event is triggered by the arrival of a reservation message m at interface i of AS v at time $\sigma.time$ at the source (but not at the start) or on the way to the start of the reservation and there is no valid reservation in v's reservation map yet. Using the event template *RES* it is instantiated as follows:

$$\begin{aligned}&\mathit{FWD}(\gamma, m, m', v, i, t) = \\ &\ \mathit{RES}(\gamma, m, m', v, i, t, (\mathit{atSrc} \wedge \neg \mathit{atSrt}) \vee \mathit{onWay}, \\ &\qquad \mathit{isFwd}, \neg \mathit{isRsvd}, \mathit{forward}, \mathit{send}, \mathit{save})\end{aligned}$$

where the parameter *updMsg* is instantiated by the function *forward*

$$\begin{aligned}&\mathit{forward}(m \in \mathcal{M}_R, \mathit{res} \in \mathit{ResMap}, \delta \in]0;1[, t \in \mathbb{N}) = \\ &\quad m(\!|\ \mathit{ptr} := \mathit{ptr} + 1\ |\!).\end{aligned}$$

The parameter *updBuf* is instantiated by the function *send* which models the sending of the processed message m' by removing the received message

m from v's buffer at interface i and adding m' to buffer i' of the next AS v'.

$$
\begin{aligned}
& \mathit{send}(v \in V, i \in I, \mathit{buf} \in \mathit{Buff}, m \in \mathcal{M}, m' \in \mathcal{M}) = \\
& \quad \textbf{let} \\
& \quad (\!| \, i', v', e' \, |\!) = \mathit{cur}(m') \\
& \quad \textbf{in} \\
& \quad \mathit{buf}\begin{pmatrix} (v,i) & \mapsto & \mathit{buf}(v,i) \setminus \{m\} \\ (v',i') & \mapsto & \mathit{buf}(v',i') \cup \{m'\} \end{pmatrix}
\end{aligned}
$$

The parameter *updRes* is instantiated by the function *save*, which writes a new entry derived from the information given by message m' in v's reservation map at entry $(\mathit{src}(m'), m'.\mathit{id})$.

$$
\begin{aligned}
& \mathit{save}(v \in V, \mathit{resM} \in \mathit{ResMap}, m' \in \mathcal{M}_R) = \\
& \quad \textbf{let} \\
& \quad \mathit{finBW} = \min(m'.\mathit{maxBW}, \min(m'.\mathit{accBW})) \\
& \quad \mathit{vrs}' = (\!| \, \mathit{minBW} := m'.\mathit{minBW}; \\
& \qquad\qquad \mathit{maxBW} := m'.\mathit{maxBW}; \\
& \qquad\qquad \mathit{idBW} := m'.\mathit{accBW}.[m'.\mathit{ptr} - 1].\mathit{idBW}; \\
& \qquad\qquad \mathit{resBW} := \mathit{finBW}; \\
& \qquad\qquad \mathit{expT} := m'.\mathit{expT} \, |\!) \\
& \quad \mathit{vrsM}' = \mathit{resM}(v, \mathit{src}(m'), m'.\mathit{id}).\mathit{vrs}\left(m'.\mathit{idx} \mapsto \mathit{vrs}'\right) \\
& \quad \mathit{res}' = (\!| \, \mathit{path} := m'.\mathit{path}; \\
& \qquad\qquad \mathit{ptr} := m'.\mathit{ptr}; \\
& \qquad\qquad \mathit{first} := m'.\mathit{first}; \\
& \qquad\qquad \mathit{last} := m'.\mathit{last}; \\
& \qquad\qquad \mathit{vrs} := \mathit{vrsM}' \, |\!) \\
& \quad \textbf{in} \\
& \quad \mathit{resM}\left((v, \mathit{src}(m'), m'.\mathit{id}) \mapsto \mathit{res}'\right).
\end{aligned}
$$

Note, that $m'.\mathit{accBW}.[m'.ptr - 1].\mathit{idBW}$ is defined as $\perp$ if the field *accBW* is the empty list *nil*.

Computation event: This event is triggered by the arrival of a reservation message m at interface i of the start AS v at time $\sigma.\mathit{time}$ at the start and on the path before the end of the reservation and if there is no valid reservation in *v*'s reservation map yet. Using the event template *RES* it is instantiated as follows:

$$\begin{aligned}&\mathit{CMP}(\gamma, m, m', v, i, t) = \\ &\ \mathit{RES}(\gamma, m, m', v, i, t, \mathit{atSrt} \vee \mathit{onPth}, \\ &\qquad \mathit{isFwd}, \mathit{notRsvd}, \mathit{process}, \mathit{send}, \mathit{save})\end{aligned}$$

$$\begin{aligned}&\mathit{process}(m \in \mathcal{M}_R, \mathit{res} \in \mathit{ResMap}, \delta \in]0;1[, t \in \mathbb{N}) = \\ &\quad \mathit{forward}\,(\mathit{compute}(m, \mathit{res}, \delta))\end{aligned}$$

where the function *compute* executes the N-Tube computations *avail* and *ideal* at *v* and adds them to *m*'s field *accBW*:

$$\begin{aligned}&\mathit{compute}(m \in \mathcal{M}_R, \mathit{res} \in \mathit{ResMap}, \delta \in]0;1[, t \in \mathbb{N}) = \\ &\textbf{let} \\ &\ \mathit{newBW} = (\!|\ \mathit{avBW} := \mathit{avail}(m, \mathit{res}, \delta, t); \\ &\qquad\qquad \mathit{idBW} := \mathit{ideal}(m, \mathit{res}, \delta, t)\ |\!) \\ &\textbf{in} \\ &\ m(\!|\ \mathit{accBW} := \mathit{newBW}\ \#\ m.\mathit{accBW}\ |\!).\end{aligned}$$

The parameter *updRes* is initiated by the function *save* as defined above. Note that there is no N-Tube computation necessary since m' contains the values of the *avail* and *ideal* computation due to the function *process*.

Turn event: This event is triggered by the arrival of a message *m* at interface *i* of AS *v* at time $\sigma.\mathit{time}$ at the end of the reservation and if there is no valid reservation in *v*'s reservation map yet. Using the event template *RES* it is instantiated as follows:

$$\begin{aligned}&\mathit{TRN}(\gamma, m, m', v, i, t) = \\ &\ \mathit{RES}(\gamma, m, m', v, i, t, \mathit{atEnd} \\ &\qquad \mathit{isFwd}, \mathit{notRsvd}, \mathit{turn}, \mathit{send}, \mathit{save})\end{aligned}$$

The parameter *updMsg* is instantiated by the function *turn* which is similar to the function *process* but decreases ptr using the function *backward*.

$$\begin{aligned}&\mathit{turn}(m \in \mathcal{M}_R, \mathit{res} \in \mathit{ResMap}, \delta \in]0;1[, t \in \mathbb{N}) = \\ &\quad \mathit{backward}\,(\mathit{compute}(m, \mathit{res}, \delta, t))\end{aligned}$$

The function *backward* does the same as the function *forward* except that it decrements the pointer ptr.

$$\begin{aligned}&\mathit{backward}(m \in \mathcal{M}_R, res \in \mathit{ResMap}, \delta \in]0;1[, t \in \mathbb{N}) =\\&\quad m(\!|\, ptr := ptr - 1 \,|\!)\end{aligned}$$

Update event: This event is triggered by the arrival of a reservation message m at interface i of an AS v on the path (not the end or the start) of the reservation at time $\sigma.\mathit{time}$, i.e. the message traverses the path backwards. Using the event template *RES* it is instantiated as follows:

$$\begin{aligned}&\mathit{UPT}(\gamma, m, m', v, i, t) =\\&\mathit{RES}(\gamma, m, m', v, i, t, \mathit{onPth} \vee (\mathit{atSrt} \wedge \neg \mathit{atSrc}),\\&\qquad \mathit{isBwd}, \mathit{isRsvd} \vee \mathit{isMrkd}, \mathit{backward}, \mathit{send}, \mathit{save})\end{aligned}$$

Backward event: This event is triggered by the arrival of a reservation message m at interface i of an AS v on the way back between start and source at time $\sigma.\mathit{time}$. Using the event template *RES* it is instantiated as follows:

$$\begin{aligned}&\mathit{BWD}(\gamma, m, m', v, i, t) =\\&\mathit{RES}(\gamma, m, m', v, i, t, \mathit{onWay},\\&\qquad \mathit{isBwd}, \mathit{isRsvd} \vee \mathit{isMrkd}, \mathit{backward}, \mathit{send}, \mathit{save})\end{aligned}$$

Finish event: This event is triggered by the arrival of a reservation message m at interface i of the source AS v of the path at time $\sigma.\mathit{time}$, i.e., the message completes its round-trip, the source updates its reservation map res with the final result and drops the message. Using the event template *RES* it is instantiated as follows:

$$\begin{aligned}&\mathit{FIN}(\gamma, m, m', v, i, t) =\\&\mathit{RES}(\gamma, m, m', v, i, t, \mathit{atSrc}\\&\qquad \mathit{isBwd}, \mathit{isRsvd} \vee \mathit{isMrkd}, \mathit{nothing}, \mathit{drop}, \mathit{save})\end{aligned}$$

The parameter *updMsg* is instantiated by the function *nothing*

$$\mathit{nothing}(m \in \mathcal{M}, res \in \mathit{ResMap}, \delta \in]0;1[, t \in \mathbb{N}) = m$$

and the parameter *updBuf* is instantiated by the function *drop*

$$\begin{aligned}&\mathit{drop}(v \in V, i \in I, \mathit{buf} \in \mathit{Buff}, m, m' \in \mathcal{M}) = \\ &\quad \mathit{buf}\,((v,i) \mapsto \mathit{buf}(v,i) \setminus \{m\})\end{aligned}$$

A.9 N-Tube's Bandwidth Allocation Computation

Available Bandwidth Computation The function *avail* computes for message m how much bandwidth is *available* on the egress link at interface e of AS v as follows. First, resM'_v is obtained from *resM* by removing the reservation corresponding to m. Second, resM'_v is obtained by extracting the reservations that go trough AS v. Finally, *avail* subtracts the aggregated *allocated bandwidth* of all currently valid reservations with the same egress interface e from the link's total capacity $\mathit{cap}(x,e)$ and multiplies the result by the parameter δ to obtain the remaining bandwidth. Multiplying with $0 < \delta < 1$ guarantees that some bandwidth is always available for subsequent reservation requests.

$$\begin{aligned}&\mathit{avail}(m, \mathit{resM}, \delta, t) = \\ &\ \textbf{let} \\ &\quad (\!|\, i, v, e \,|\!) = \mathit{cur}(m) \\ &\quad \mathit{resM}' = \mathit{resM}\,((v, \mathit{src}(m), m.\mathit{id}) \mapsto \bot) \\ &\quad \mathit{resM}'_v = \mathit{filter}(\mathit{resM}', v) \\ &\ \textbf{in} \\ &\ \delta \cdot \left(\mathit{cap}(v,e) - \sum\nolimits_{\substack{r \in \mathit{rng}(\mathit{resM}'_v): \\ \mathit{resEg}(r) = e}} \mathit{allocBW}(r.\mathit{vrs}, t)\right)\end{aligned}$$

Given an AS v and a reservation *resM* the function *filter* restricts *resM* to the reservations that go through v.

$$\begin{aligned}&\mathit{filter}(\mathit{resM}, v) = \\ &\ \lambda(s', \mathit{id}'). \\ &\ \textbf{let}\ r = \mathit{resM}(v, s', \mathit{id}') \\ &\ \textbf{in}\ \big(\textbf{if}\ r.\mathit{first} \le r.\mathit{ptr} \le r.\mathit{last}\ \textbf{then}\ \mathit{resM}(v, s', \mathit{id}')\ \textbf{else}\ \bot\big)\end{aligned}$$

Given the current time t, a *currently valid* version *vrs* is not *expired*, i.e., $\mathit{vrs}.\mathit{expT} \ge t$, and *successful*, i.e., $\mathit{vrs}.\mathit{minBW} \le \mathit{vrs}.\mathit{resBW}$. The reservation's bandwidth *allocation* are computed by the function *allocBW* and is defined as the maximum of its currently valid versions' $resBW$.[1]

$$\begin{aligned}&allocBW(vrsM, t) = \\ &\quad \max_{\substack{vrs\in\\ rng(vrsM)}} \{vrs.resBW \mid vrs.minBW \leq vrs.resBW \land vrs.expT \geq t\}\end{aligned}$$

The maximum is taken, since the source can send traffic using any existing version of its reservations. Hence, this computation guarantees that, in the worst-case, enough bandwidth is available.

Analogously, the function *demBW* computes reservation's bandwidth *demand* and is defined as the maximum of its currently valid versions' $maxBW$.

$$\begin{aligned}&demBW(vrsM, t) = \\ &\quad \max_{\substack{vrs\in\\ rng(vrsM)}} \{vrs.maxBW \mid vrs.minBW \leq vrs.resBW \land vrs.expT \geq t\}\end{aligned}$$

Ideal Bandwidth Computation Given a message m, the function *ideal* computes how the adjusted capacity $\delta \cdot cap(v, e)$ of the egress link e of AS v is shared in a so-called *bounded tube-fair* manner among all the existing reservations at AS v with the same egress link e. First, $resM'$ is obtained from $resM$ by removing all existing versions and adding a new version corresponding to m. Removing previous versions of the reservation guarantees that the result of the *ideal* computation is not influenced by versions that are still valid and therefore simulates the ideal state where only versions of the reservation exist which correspond to m. Second, $resM'_v$ is obtained by extracting the reservations that go trough AS v. Finally, *ideal* first proportionally splits the egress link's adjusted capacity between each ingress link by multiplying with *tubeRatio*, partitions the result between reservations starting and traversing the ingress link i by multiplying with *linkRatio*, and splits the result proportionally between all remaining reservations requests by multiplying with *reqRatio*.

[1]Note that $\max \emptyset = 0$.

```
$ideal(m, resM, \delta, t) =$
 let
  $s = src(m)$
  $id = m.id$
  $\llparenthesis\, i, v, e \,\rrparenthesis = cur(m)$
  $vrs' = \llparenthesis\, minBW := m.minBW;$
            $maxBW := m.maxBW;$
            $idBW := m.accBW.[m.ptr - 1].idBW;$
            $resBW := m.minBW;$
            $expT := m.expT \,\rrparenthesis$
  $vrsM' = \emptyset\big(m.idx \mapsto vrs'\big)$
  $res' = \llparenthesis\, path := m.path;$
            $ptr := m.ptr;$
            $first := m.first;$
            $last := m.last;$
            $vrs := vrsM' \,\rrparenthesis$
  $resM' = resM\big((v, s, id) \mapsto vrsM'\big)$
  $resM'_v = filter(resM', v)$
  if $(m.first < m.ptr)$
  then $reqRatio = reqRatio_{transit}(v, s, id, i, resM'_v, t)$
         $linkRatio = linkRatio_{transit}(v, i, resM'_v, t)$
  else $inRatio = reqRatio_{start}(v, s, id, i, resM'_v, t)$
         $linkatio = linkRatio_{start}(v, i, resM'_v, t)$
 in
  $reqRatio \cdot linkRatio \cdot tubeRatio(v, i, e, resM'_v, t) \cdot \delta \cdot cap(v, e)$
```

Tube Ratio: The *tube ratio* between an ingress interface i and an egress interface e is computed as the ratio of the *bounded tube demand* between i and e, given by $\min\{cap(v,i), tubeDem(i,e)\}$, and the aggregated bounded tube demands at e.

$$\begin{aligned}&\mathit{tubeRatio}(v, i, e, \mathit{resM}, t) = \\ &\frac{\min\{\mathit{cap}(v, i), \mathit{tubeDem}(v, i, e, \mathit{resM}, t)\}}{\sum_{i' \in I} \min\{\mathit{cap}(v, i'), \mathit{tubeDem}(v, i', e, \mathit{resM}, t)\}}.\end{aligned}$$

Taking the minimum with respect to the corresponding ingress link's capacity guarantees that its respective portion of tube demand compared to the other ingress links' tube demands is always bounded. This prevents that the bandwidth reserved for other ingress links will be reduced ad infinitum.

The *tube demand* between an ingress interface i and an egress interface e aggregates their *adjusted requested demands*

$$\begin{aligned}&\mathit{tubeDem}(v, i, e, \mathit{resM}, t) = \\ &\sum_{\substack{r \in \mathit{rng}(\mathit{resM}): \\ \mathit{resIn}(r)=i \\ \mathit{resEg}(r)=e}} \mathit{adjReqDem}(v, r, i, e, \mathit{resM}, t).\end{aligned}$$

A source can demand more than the ingress and egress links' capacities allow. To account for that, the *adjusted requested demand* of a reservation r is derived from its *requested demand*, by multiplying the latter with the minimum of two *scaling factors*

$$\begin{aligned}&\mathit{adjReqDem}(v, r, \mathit{resM}, t) = \\ &\mathbf{let} \\ &\quad s = \mathit{resSr}(r) \\ &\quad i = \mathit{resIn}(r) \\ &\quad e = \mathit{resEg}(r) \\ &\mathbf{in} \\ &\quad \min\{\mathit{inScalFctr}(v, s, i, \mathit{resM}, t), \mathit{egScalFctr}(v, s, e, \mathit{resM}, t)\} \\ &\qquad\qquad \cdot \mathit{reqDem}(v, r, i, e, t).\end{aligned}$$

Analogously to the *allocated bandwidth allocBW* in *avail*, the *requested demand* of a reservation r is the maximum of its *demanded bandwidth demBW*

$$\begin{aligned}&\mathit{reqDem}(v, r, i, e, t) = \\ &\min\{\mathit{cap}(v, i), \mathit{cap}(v, e), \mathit{demBW}(r.\mathit{vrs}, t)\}.\end{aligned}$$

Note that, the *requested demand* is bounded by the ingress and egress links' capacities. This avoids that s reserves more bandwidth in one request than physically possible, i.e., this guarantees that

$$reqDem(v, r, i, e, t) \leq \min\{cap(v, i), cap(v, e)\}.$$

Analogously to the *allocated bandwidth allocBW* in *avail*, the *requested demand* of a reservation r is the maximum of its *demanded bandwidth demBW* .

$$demBW(vrsM, t) = \max_{\substack{vrs\in \\ rng(vrsM)}} \{vrs.maxBW \mid vrs.minBW \leq vrs.resBW \wedge vrs.expT \geq t\}$$

The maximum is taken, since the source can send traffic using any existing version of its reservations. Hence, this computation guarantees that, in the worst-case, enough bandwidth is available.

However, it is possible that s demands less than the capacity of an ingress (or egress) link in each of its requests, but the aggregate of all its demands might still exceed the link's capacity. To adjust the *requested demands,* we multiply them with the minimum of the corresponding *ingress* and the *egress scaling factor*. We compute the *egress scaling factor* on the egress link at e for s as the source's proportion of the total *egress demand* bounded by the egress link's capacity, given by the function

$$egScalFctr(v, s, e, resM, t) = \frac{\min\left(cap(v, e), egDem(v, s, e, resM, t)\right)}{egDem(v, s, e, resM, t)}.$$

Analogously, we define the *ingress scaling factor* on the egress link at e for s

$$inScalFctr(v, s, i, resM, t) = \frac{\min\left(cap(v, i), inDem(v, s, i, resM, t)\right)}{inDem(v, s, i, resM, t)}.$$

The *egress demand* of s on e is defined as the aggregate over its *requested demands* with egress interface e,

$$egDem(v, s, e, resM, t) = \sum_{\substack{r'\in rng(resM): \\ resSr(r')=s \\ resEg(r')=e}} reqDem(v, r', resIn(r'), e, t).$$

Analogously, we compute the source's *ingress scaling factor* on the ingress interface i,

$$\begin{aligned}&inDem(v,s,i,resM,t) = \\ &\sum\nolimits_{\substack{r'\in rng(resM):\\ resSr(r')=s\\ resIn(r')=i}} reqDem(v,r',i,resEg(r'),t).\end{aligned}$$

Link Ratio: If v is a transit AS on m's path, i.e., $m.first < m.ptr$ ($\leq m.last$), then the *link ratio* between an ingress interface i and an egress interface e is computed as the ratio of the *bounded transit demand* between i and e, given by $\min\{cap(v,i), transitDem(i,e)\}$, and the sum of bounded start and bounded transit demand

$$\begin{aligned}&linkRatio_{transit}(v,i,resM,t) = \\ &\textbf{let} \\ &\;stDem = startDem(v,i,resM,t) \\ &\;trDem = transitDem(v,i,resM,t) \\ &\textbf{in} \\ &\;\frac{\min\{cap(v,i), trDem\}}{\min\{cap(v,i), stDem\} + \min\{cap(v,i), trDem\}}.\end{aligned}$$

If v is the first AS on m's path, i.e., $m.first = m.ptr$, then the *link ratio* between an ingress interface i and an egress interface e is computed analogously with *startDem* in the nominator instead of *transitDem*

$$\begin{aligned}&linkRatio_{start}(v,i,resM,t) = \\ &\textbf{let} \\ &\;stDem = startDem(v,i,resM,t) \\ &\;trDem = transitDem(v,i,resM,t) \\ &\textbf{in} \\ &\;\frac{\min\{cap(v,i), stDem\}}{\min\{cap(v,i), stDem\} + \min\{cap(v,i), trDem\}}.\end{aligned}$$

Taking the minimum with respect to ingress link's capacity guarantees that its respective portion for transit demand compared to demands of reservations starting at i is always bounded. This prevents that the bandwidth allocated for traversing reservations can be reduced ad infinitum by excessive reservations starting at link i and vice-versa. Here the *transit demand* at an ingress interface i *adjusted request demands* of traversing reservations,

$$\mathit{transitDem}(v, i, \mathit{resM}, t) = \sum_{\substack{r \in \mathit{rng}(\mathit{resM}):\\ \mathit{resIn}(r)=i\\ r.\mathit{first}<r.\mathit{ptr}}} \mathit{adjIdDem}(v, r, i, \mathit{resM}, t).$$

Analogously, the *startDem* at an ingress interface i *adjusted request demands* of starting reservations,

$$\mathit{startDem}(v, i, \mathit{resM}, t) = \sum_{\substack{r \in \mathit{rng}(\mathit{resM}):\\ \mathit{resIn}(r)=i\\ r.\mathit{first}=r.\mathit{ptr}}} \mathit{adjIdDem}(v, r, \mathit{resM}, t).$$

The *adjusted previous ideal bandwidth allocations* of a reservation r is similarly defined to *adjReqDem* the *previous ideal bandwidth allocation* with the *egress scaling factors* of the corresponding source AS and egress link

$$\begin{aligned}
&\mathit{adjIdDem}(v, r, \mathit{resM}, t) = \\
&\textbf{let} \\
&\quad s = \mathit{resSr}(r) \\
&\quad i = \mathit{resIn}(r) \\
&\quad e = \mathit{resEg}(r) \\
&\textbf{in} \\
&\mathit{egScalFctr}(v, s, e, \mathit{resM}, t) \cdot \min\{\mathit{cap}(v, i), \mathit{cap}(v, e), \mathit{idBW}(r.\mathit{vrs}, t)\}.
\end{aligned}$$

The *previous ideal bandwidth allocation* of a reservation's version map is similarly defined to the *allocated bandwidth* by maximizing over the field *idBW* instead of *resBW*.

$$\mathit{idBW}(\mathit{vrsM}, t) = \max_{\substack{\mathit{vrs} \in\\ \mathit{rng}(\mathit{vrsM})}} \{\mathit{vrs}.\mathit{idBW} \mid \mathit{vrs}.\mathit{minBW} \leq \mathit{vrs}.\mathit{resBW} \land \mathit{vrs}.\mathit{expT} \geq t\}.$$

Request Ratio: If v is a transit AS on m's path, i.e., $m.\mathit{first} < m.\mathit{ptr}$ ($\leq m.\mathit{last}$), then the *request ratio* of a reservation identified by (s, id) at ingress interface i is the ratio between its *adjusted ideal bandwidth allocation* (provided by the predecessor on the reservation's path) and the *transit* *demand* at interface *i*

$$reqRatio_{transit}(v, s, id, i, resM, t) = \frac{adjIdDem(v, resM(v, s, id), resM, t)}{transitDem(v, i, resM, t)}.$$

Similarly, in case v is the first AS on m's path, i.e., $m.first = m.ptr$, we the *request ratio* by

$$reqRatio_{start}(v, s, id, i, resM, t) = \frac{adjIdDem(v, resM(v, s, id), resM, t)}{startDem(v, i, resM, t)}.$$

A.10 Deletion Events

The two events triggered by a arrival of a deletion message are given by the following event template:

```
$DEL(m, m' \in \mathcal{M}_D, v \in V, i \in I, t \in \mathbb{N},$
    $Loc : \mathcal{M} \rightarrow \mathbb{B},$
    $updBuf : V \times I \times Buff \times \mathcal{M} \times \mathcal{M} \rightarrow Buff$
$= \{(\sigma, \sigma') \mid$
    - guards -
    $m \in \sigma.buf(v, i) \wedge$
    $\sigma.time = t \wedge$
    $PathCheck(m.path) \wedge$
    $ResMapCheck_D(m, v, \sigma.res) \wedge$
    $Loc(delMsg(m)) \wedge$
    $m.path[m.ptr].as = v \wedge$
    $m.path[m.ptr].inI = i \wedge$
    $m' = m(\!| ptr := ptr + 1 |\!) \wedge$
    - actions -
    $\sigma'.buf = updBuf(v, i, \sigma.buf, delMsg(m), delMsg(m')) \wedge$
    $\sigma'.res = remove(v, \sigma.res, m) \}$
```

The function *remove* overwrites an entry given by a message m in the reservation map res of AS v

$$\begin{aligned}
&remove(v \in V, res \in ResMap, m \in \mathcal{M}_D) = \\
&\ \textbf{let} \\
&\ \ delVrs = res(v, src(m), m.id).vrs\,(m.idx \mapsto \bot) \\
&\ \textbf{in} \\
&\ \ res\big((v, src(m), m.id) \mapsto \llparenthesis\, vrs := delVrs\, \rrparenthesis\big)
\end{aligned}$$

Note that the following parts were changed compared to the reservation message arrival template *RES*:

- Environment γ is not needed, since there is no N-Tube computation in deletion events.
- Predicate *Dir* instantiated is replaced with the first two conjuncts of predicate *isFwd*, since deletion messages only travels the path forward once and the *accBW* field is not needed in deletion messages.
- Predicate *Rsvd* is not needed, since if there is no version corresponding to m, then function *remove* does not change it.
- Function *updMsg* is replaced by the term of function *forward*.
- Function *updRes* is initiated with the function *remove*
- Predicate *ResMsgCheck* is not necessary since it validates fields of reservation messages that are not part of deletion messages.

There are two instantiations of the *DEL* event template:

Remove event: This event is triggered by the arrival of a deletion message m at interface i of AS v which is the source or on the path (but not the destination) at time $\sigma.time$. Using the event template *DEL* it is instantiated by

$$\begin{aligned}
&RMV(m, m', v, i, t) = \\
&\ DEL(m, m', v, i, t, prePth, send)
\end{aligned}$$

with the path predicate $onPth : \mathcal{M} \to \mathbb{B}$

$$\begin{aligned}
&prePth(m \in \mathcal{M}) = \\
&\ m.ptr < length(m.path).
\end{aligned}$$

Destination event: This event is triggered by the arrival of a deletion message m at interface i of the destination AS v of the path at time $\sigma.time$. Using the event template DEL it is instantiated by

$$\begin{aligned}&DST(m, m', v, i, t) = \\ &\ DEL(m, m', v, i, t, atDst, drop).\end{aligned}$$

with the path predicate $atDst : \mathcal{M} \to \mathbb{B}$

$$\begin{aligned}&atDst(m \in \mathcal{M}) = \\ &\ m.ptr = length(m.path)\end{aligned}$$

A.11 Drop Events

This event is triggered by the arrival of a deletion or reservation message m at interface i of the source AS v of the path at time $\sigma.time$

$$\begin{aligned}&DRP(m \in \mathcal{M}, v \in V, i \in I, t \in \mathbb{N}) = \{(\sigma, \sigma') \mid \\ &\quad \texttt{- guards -} \\ &\quad m \in \sigma.buf(v, i) \wedge \\ &\quad \sigma.time = t \wedge \\ &\quad \texttt{- actions -} \\ &\quad \sigma'.buf = \sigma.buf(\ (v, i) \mapsto \sigma.buf(v, i) \setminus \{m\}\ )\ \}\end{aligned}$$

A.12 Attack Events

The attack events are given as described before. However, in Theorem 23, we assume a stronger attacker model by weakening the guards of the ATK event

$$\begin{aligned}&ATK(m \in \mathcal{M}, v \in V, i \in I) = \{(\sigma, \sigma') \mid \\ &\quad \texttt{- guards -} \\ &\quad m \in \sigma.kwl \wedge \\ &\quad \texttt{- actions -} \\ &\quad \sigma'.buf = \sigma.buf\,((v, i) \mapsto \sigma.buf(v, i) \cup \{m\})\ \}.\end{aligned}$$

In the previously defined ATK event we restrict by the additional guard $((a, e), (v, i)) \in E$ that v has to be honest and connected with a link to a malicious AS $a \in M$. Note, however, proving properties with this stronger attacker model does imply these properties also for the weaker one.

A.13 Executions

Monotonicity Given the transition relation Δ defined in Section 3.5 for any execution $\pi \in \mathfrak{E}$ is the property of *time-monotonicity*, i.e.,

Lemma 3.

$$\forall n, \tilde{n} \in \mathbb{N}.\ n \leq \tilde{n} \Rightarrow \sigma_n.time \leq \sigma_{\tilde{n}}.time$$

Proof. It's sufficient to show:

$$\forall n \in \mathbb{N}.\ \sigma_n.time \leq \sigma_{n+1}.time$$

Given $n \in \mathbb{N}$. By case distinction on λ_n:

- $TCK(\tilde{t})$: By the event's action it holds $\sigma_{n+1}.time = \sigma_n.time + 1 > \sigma_n.time$.
- All other event's actions keep the time unchanged, i.e., $\sigma_n.time = \sigma_{n+1}.time$.

□

Time-progress The global time infinitely progresses, i.e.,

$$\forall t \in \mathbb{N}\ \exists n \in \mathbb{N}.\ \sigma_n.time \geq t, \tag{TP}$$

more well-known as the property of *zeno-freeness*.

This is reasonable to assume, since it is equivalent to the assumption that in a realistic execution there are only finitely many ATK and CRT events at any given point in time

$$\begin{aligned}
&\forall t \in \mathbb{N}\ \exists B \in \mathbb{N}.\\
&|\{n \in \mathbb{N} \mid \lambda_n.time = t \wedge \lambda_n.evt = ATK\}| \leq B\ \wedge\\
&|\{n \in \mathbb{N} \mid \lambda_n.time = t \wedge \lambda_n.evt = CRT\}| \leq B.
\end{aligned}$$

Message-Progress: All messages in the buffers of honest ASes are processed in at most time *bufT*

$$\begin{aligned}
&\forall n \in \mathbb{N}, v \in H, i \in I, m \in \sigma_n.buf(v,i).\\
&\quad \exists \tilde{n} > n.\ m \notin \sigma_{\tilde{n}}.buf(v,i)\ \wedge\ \sigma_{\tilde{n}}.time - \sigma_n.time \leq bufT.
\end{aligned} \tag{MP}$$

Distinct-Pointers: W.l.o.g., we assume (DP) that all messages m their field *first* is positive which is given as Assumption DP

$$\begin{aligned}&\forall n \in \mathbb{N}, v \in V, i \in I, m \in \mathcal{M}.\\ &\quad m \in \sigma_n.buf(v,i) \Rightarrow 0 < m.first\end{aligned} \tag{DP}$$

Note that we assumed $m.first < m.last$. Adding this assumption does not change any of the properties we prove but avoids considering additional corner-cases.

Successful Reservation We say *an honest source $s \in H$ makes a successful reservation confirmed by the message $m \in \mathcal{M}_R$ at time t* if the following three conditions hold:

Honest Path: m's path only contains honest ASes.

$$nodes(m) \subseteq H. \tag{HP}$$

Confirmation: s confirms m at time t with sufficient bandwidth

$$\exists n \in \mathbb{N}, i \in I.\ \lambda_n = FIN(m,s,i,t) \wedge finBW(m) \geq m.minBW. \tag{CF}$$

No deletion: There is no deletion event matching the reservation $(src(m), m.id)$ and version $m.idx$ before m expires

$$\begin{aligned}&\forall \hat{n}, n_c \in \mathbb{N}, \hat{v} \in V, \hat{\imath}, i \in I, \hat{m} \in \mathcal{M}_D, m_c \in \mathcal{M}_R, \hat{t}, t_c \in \mathbb{N}.\\ &\lambda_{n_c} = CRT_R(m_c, s, i, t_c) \wedge m_c \approx m\ \wedge\\ &\sigma_{\hat{n}}.time \in\ ]t_c; m.expT]\ \wedge\\ &\big(\lambda_{\hat{n}} = RMV(\hat{v}, \hat{\imath}, \hat{m}, \hat{t})\ \vee\ \lambda_{\hat{n}} = DST(\hat{v}, \hat{\imath}, \hat{m}, \hat{t})\big)\\ &\quad \Rightarrow m_c \not\approx \hat{m}.\end{aligned} \tag{nDE}$$

Constant Demands We model "constant bandwidth demands" using a function

$$\mathcal{D} : V \times \mathbb{N} \rightharpoonup_f \mathcal{M}_R$$

such that $\mathcal{D}(s, id) = m$ implies $src(m) = s$ and $m.id = id$, and $\mathcal{D}(s, id).minBW = 0$.

We say that a *reservation message m corresponds to $\mathcal{D}$* if $(src(m), m.id) \in supp(\mathcal{D})$ and m coincides with $\mathcal{D}(src(m), m.id)$ on all fields except *ptr*, *expT*, and *accBW*.

$$\begin{aligned}
&m \vdash \mathcal{D} \;:\Leftrightarrow \\
&\quad \mathcal{D}(src(m), m.id).path = m.path \;\wedge \\
&\quad \mathcal{D}(src(m), m.id).first = m.first \;\wedge \\
&\quad \mathcal{D}(src(m), m.id).last = m.last \;\wedge \\
&\quad \mathcal{D}(src(m), m.id).minBW = m.minBW \;\wedge \\
&\quad \mathcal{D}(src(m), m.id).maxBW = m.maxBW
\end{aligned}$$

We say that a *reservation r identified by (s, id) corresponds to $\mathcal{D}$ at time t* if $(s, id) \in supp(\mathcal{D})$ and r coincides with $\mathcal{D}(s, id)$ on all fields except *ptr*, *expT*, for all its versions and *accBW*.

$$\begin{aligned}
&r \vdash \mathcal{D} \;:\Leftrightarrow \\
&\quad (s, id) \in supp(\mathcal{D}) \;\wedge \\
&\quad \mathcal{D}(s, id).path = r.path \;\wedge \\
&\quad \mathcal{D}(s, id).first = r.first \;\wedge \\
&\quad \mathcal{D}(s, id).last = r.last \;\wedge \\
&\quad \forall idx \in \mathbb{N}. \\
&\qquad r.vrs(idx) = \bot \;\vee \\
&\qquad r.vrs(idx).expT < t \;\vee \\
&\qquad \mathcal{D}(s, id).minBW = r.vrs(idx).minBW \;\wedge \\
&\qquad \mathcal{D}(s, id).maxBW = r.vrs(idx).maxBW\;)
\end{aligned}$$

We say that a *reservation map res_v corresponds to $\mathcal{D}$ at time t at AS v* if all its reservation correspond to $\mathcal{D}$ at time t

$$\forall r \in rng(res_v)\,.\; r \vdash \mathcal{D}$$

We say that a *state σ corresponds to $\mathcal{D}$ at time t* if all for any honest ASes $v \in H$ its *reservation map res_v corresponds to $\mathcal{D}$ at time t at AS v*

$$\forall v \in H, r \in rng(\sigma.res_v)\,.\; r \vdash \mathcal{D}$$

For the rest of this section we fix two time points $t_0, t_1 \in \mathbb{N}$ such that $t_1 - t_0 \geq 2maxT$. We say an execution $\pi \in \mathfrak{E}$ *has constant demands $\mathcal{D}$ between t_0 and t_1* if

Successful Requests: for all $(s, id) \in supp(\mathcal{D})$ the source AS s has successfully made a reservation confirmed by a message m corresponding to $\mathcal{D}(s, id)$ before time t_0

$$\begin{aligned}
&\forall m \in rng(\mathcal{D}), evt \in \{CMP, TRN, UPT\}, v \in sgmt(m).\\
&\ \exists \tilde{n} \in \mathbb{N}, \tilde{m}, \tilde{m}' \in \mathcal{M}_R, \tilde{i} \in I, \tilde{t} \in \mathbb{N}.\\
&\ \ \sigma_{\tilde{n}}.time \leq t_0 \ \wedge \ \tilde{m} \vdash \mathcal{D} \ \wedge \ \lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', v, \tilde{i}, \tilde{t}) \ \wedge \ \tilde{m}.expT > t_0,
\end{aligned}$$

Successful Renewal: and successfully renews this reservation without any gaps until t_1,

$$\begin{aligned}
&\forall \tilde{n} \in \mathbb{N}, evt \in \{CMP, TRN, UPT\}.\ \sigma_{\tilde{n}}.time \in [t_0, t_1] \Rightarrow\\
&\ \forall \tilde{m}, \tilde{m}' \in \mathcal{M}_R, \tilde{v} \in H, \tilde{i} \in I, \tilde{t} \in \mathbb{N}.\\
&\ \lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t}) \ \wedge \ \tilde{m} \vdash \mathcal{D} \ \Rightarrow\\
&\ \ \exists \hat{n} \in \mathbb{N}, \hat{m}, \hat{m}' \in \mathcal{M}_R, \hat{v} \in sgmt(\hat{m}), \hat{i} \in I, \hat{t} \in \mathbb{N}.\\
&\ \ \ \sigma_{\hat{n}}.time \in [\tilde{t}, \tilde{m}.expT] \ \wedge \ \lambda_{\hat{n}} = evt(\hat{m}, \hat{m}', \hat{v}, \hat{i}, \hat{t}) \wedge \ \hat{m}.expT > \tilde{m}.expT.
\end{aligned}$$

Constant Demands: any reservation confirmed by a reservation message m between t_0 and t_1, corresponds to $\mathcal{D}$

$$\begin{aligned}
&\forall \tilde{n} \in \mathbb{N}, evt \in \{CMP, TRN, UPT\}.\ \sigma_{\tilde{n}}.time \in [t_0, t_1] \Rightarrow\\
&\forall \tilde{m}, \tilde{m}' \in \mathcal{M}_R, \tilde{i} \in I, \tilde{t} \in \mathbb{N}, \tilde{v} \in sgmt(\tilde{m}).\\
&\lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t}) \Rightarrow \tilde{m} \vdash \mathcal{D}
\end{aligned}$$

and similarly for attack events

$$\begin{aligned}
&\forall \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time \in [t_0, t_1] \Rightarrow\\
&\ \forall \tilde{m} \in \mathcal{M}_R, \tilde{i}, \tilde{e} \in I, \tilde{t} \in \mathbb{N}, \tilde{a} \in M, \tilde{v} \in sgmt(\tilde{m}).\\
&\ \lambda_{\tilde{n}} = ATK(\tilde{m}, \tilde{a}, \tilde{v}, \tilde{i}, \tilde{e}, \tilde{t})\\
&\ \ \Rightarrow \tilde{m} \vdash \mathcal{D} \ \wedge \ finBW(\tilde{m}) = \mathcal{D}(src(\tilde{m}), \tilde{m}.id).maxBW
\end{aligned}$$

No Deletion: there are no deletion events between t_0 and t_1 for reservations given by $supp(\mathcal{D})$.

$$\begin{aligned}
&\forall \tilde{n} \in \mathbb{N}, evt \in \{RMV, DST\}.\ \sigma_{\tilde{n}}.time \in [t_0, t_1]\\
&\forall \tilde{m} \in \mathcal{M}_D, \tilde{v} \in H, \tilde{i} \in I, \tilde{t} \in \mathbb{N}.\\
&\ \lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t}) \Rightarrow (src(\tilde{m}), \tilde{m}.id) \notin supp(\mathcal{D})
\end{aligned}$$

Appendix B
Appendix: Proofs

To not increase the appendix further with technical poofs, we just provide the most representative ones to give an idea how such poofs look like.

B.1 Event Lemmata

Lemma 4.

$$\begin{aligned}
&\forall n \in \mathbb{N}, s \in V, i \in I, m \in \mathcal{M}_R, t \in \mathbb{N}.\\
&\quad \lambda_n = CRT_R(s,i,m,t) \;\Rightarrow\\
&\quad m \in \sigma_{n+1}.buf(s,i_s) \;\land\\
&\quad PathCheck(m) \;\land\; ResMsgCheck(m, \sigma_{n+1}.time) \;\land\\
&\quad \sigma_{n+1}.time = t \;\land\\
&\quad atSrc(m) \;\land\\
&\quad isFwd(s,i_s,m) \;\land\\
&\quad ResMapCheck(\sigma_{n+1}.res, s, m) \;\land\; isMrkd(\sigma_{n+1}.res, s, m, \sigma_{n+1}.time)
\end{aligned}$$

Proof. $m \in \sigma_{n+1}.buf(s,i_s)$: follows by the first action of CRT_R, i.e.

$$\sigma_{n+1}.buf = \sigma_n.buf\,((s,i_s) \mapsto \sigma.buf(s,i_s) \cup \{m\})\,\}$$

$PathCheck(m)$: Follows by first guard of CRT_R.

$ResMsgCheck(m, \sigma_{n+1}.time)$: Follows by second guard and since the action of CRT_R does not change $\sigma_n.time$, i.e. $\sigma_n.time = \sigma_{n+1}.time$.

$ResMapCheck(\sigma_{n+1}.res, s, m)$: Follows by third guard and since the action of CRT_R does not change $\sigma_n.res$, i.e. $\sigma_n.res = \sigma_{n+1}.res$.

$\sigma_{n+1}.time = t_c$: Follows by the fourth guard and since the action of CRT_R does not change $\sigma_n.time$, i.e. $\sigma_n.time = \sigma_{n+1}.time$.

$atSrc(m)$: Follows by the sixth guard of CRT_R.

$\neg isRsvd(\sigma_{n+1}.res, s, m, \sigma_{n+1}.time)$: Follows from the seventh guard and since the actions of CRT_R do not change $\sigma_n.res$, i.e. $\sigma_n.res = \sigma_{n+1}.res$.

□

Lemma 5.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = FWD(m, m', v, i, t) \Rightarrow\\
&\quad\quad m \approx m' \;\wedge\; m'.ptr = m.ptr + 1 \;\wedge\; m'.accBW = m.accBW = nil \wedge\\
&\quad\quad isFwd(v', i', m') \wedge\\
&\quad\quad m \in \sigma_n.buf(v, i) \wedge m \notin \sigma_{n+1}.buf(v, i) \wedge m' \in \sigma_{n+1}.buf(v', i') \wedge\\
&\quad\quad PathCheck(m') \;\wedge\; ResMsgCheck(m', \sigma_{n+1}.time) \wedge\\
&\quad\quad \sigma_{n+1}.time = \sigma_n.time = t \;\wedge\\
&\quad\quad onWay(m') \vee atSrt(m') \wedge\\
&\quad\quad \forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx.\\
&\quad\quad\quad \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx) \wedge\\
&\quad\quad ResMapCheck(\sigma_{n+1}.res, v, m) \wedge isMrkd(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)
\end{aligned}$$

Proof. **First seven statements:** By event's first guard it holds $m \in \sigma_n.buf(v, i)$. By the event's guard $m' = forward(m, \sigma_n.res, \gamma.\delta)$ it follows that $m'.ptr = m.ptr + 1$ and the rest of the fields stay the same, hence, $m \approx m'$ and $m.accBW = m'.accBW$. This implies $m'.path = m.path$ and therefore

$$\begin{aligned}
m.path[m.ptr + 1].as &= m'.path[m'.ptr].as = v'\\
m.path[m.ptr + 1].as &= m'.path[m'.ptr].inI = i'.
\end{aligned}$$

This is by definition equivalent to $isFwd(v', i', m')$. $m.accBW = nil$ follows by the event's guards $isFwd(v, i, m)$ and $atSrc(m) \vee onWay(m)$. Furthermore, it follows by the definition of function *send*

$$\begin{aligned}
\sigma_{n+1}.buf(v, i) &= \sigma_n.buf(v, i) \setminus \{m\}\\
\sigma_{n+1}.buf(v', i') &= \sigma_n.buf(v', i') \cup \{m'\}.
\end{aligned}$$

and therefore $m \notin \sigma_{n+1}.buf(v, i)$, and $m' \in \sigma_{n+1}.buf(v', i')$.

$PathCheck(m')$: Follows by the second guard of FWD and $m' \approx m$, hence $m.path = m'.path$.

$ResMsgCheck(m', \sigma_{n+1}.time)$: By the third guard of FWD we have $ResMsgCheck(m, \sigma_n.time)$. The actions of FWD do not change $\sigma_n.time$, i.e. $\sigma_n.time = \sigma_{n+1}.time$. FWD's guard $m' = forward(m, \sigma_n.res, \gamma.\delta)$ only changes the field ptr, which *ResMsgCheck* does not depend on, and keeps all the other fields the same.

$\sigma_{n+1}.time = t$: Follows by the fifth guard and since the action of FWD does not change $\sigma_n.time$, i.e. $\sigma_n.time = \sigma_{n+1}.time = t$.

$onWay(m') \lor atSrt(m')$: Follows by the seventh guard of FWD for m, i.e. $0 \leq m.ptr < m.first$ and that $m'.ptr = m.ptr + 1$ and $m.first = m'.first$, given by the sixth guard of FWD. Hence, together it follows that $0 < m'.ptr \leq m'.first$ or equivalently $onWay(m') \lor atSrt(m')$.

$\neg isRsvd(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)$: Follows by FWD's action $\sigma_{n+1}.res = mark(v, \sigma_n.res, m')$ that sets

$$\sigma_{n+1}.res(v, src(m'), m'.id).vrs(m'.idx) = \bot$$

and from above that $m \approx m'$, i.e. $src(m) = src(m')$, $m'.id = m.id$, and $m'.idx = m.idx$.

Unchanged Reservations: This follows directly from the event's action only changing the reservation corresponding to $src(m)$, $m.id$, and $m.idx$, i.e.

$$\begin{gathered}\forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx. \\ \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx)\end{gathered}$$

$ResMapCheck(\sigma_{n+1}.res, v, m)$: By the event's action $\sigma_{n+1}.res = save(v, \sigma_n.res, m')$ and $m' \approx m$ for $r = \sigma_{n+1}.res(v, src(m), m.id)$ it holds that

$$\begin{aligned}&r.path = m.path\\&r.ptr = m.ptr\\&r.first = m.first\\&r.last = m.last\\&r.vrs(idx).minBW = m.minBW\\&r.vrs(idx).maxBW = m.maxBW\\&r.vrs(idx).expT = m.expT\end{aligned}$$

Hence, by definition of $ResMapCheck$ this implies $ResMapCheck(\sigma_{n+1}.res, v, m)$.

$isRsvd(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)$: In the same way this implies by definition of $isRsvd$ that $isRsvd(\sigma_{n+1}.res, v, m, m.expT)$ holds. And by the event's guards it follows $m.expT > \sigma_{time} = \sigma_{n+1}.time$ and therefore $isRsvd(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)$.

□

Lemma 6.

$$
\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\ \lambda_n = CMP(m, m', v, i, t) \ \Rightarrow\\
&\ m \approx m' \ \wedge \ m'.ptr = m.ptr + 1 \ \wedge \ m'.accBW = newBW \ \# \ m.accBW \ \wedge\\
&\ isFwd(v', i', m') \wedge\\
&\ m \in \sigma_n.buf(v, i) \wedge m \notin \sigma_{n+1}.buf(v, i) \wedge m' \in \sigma_{n+1}.buf(v', i') \wedge\\
&\ PathCheck(m') \ \wedge \ ResMsgCheck(m', \sigma_{n+1}.time) \wedge\\
&\ \sigma_{n+1}.time = \sigma_n.time = t \ \wedge\\
&\ (atSrt(m) \vee onPth(m)) \wedge \big(onPth(m') \vee atEnd(m')\big) \ \wedge\\
&\ \forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx.\\
&\quad \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx) \wedge\\
&\ ResMapCheck(\sigma_{n+1}.res, v, m) \ \wedge \ resCheck(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)
\end{aligned}
$$

with

$$
\begin{aligned}
newBW = (\!| \ & avBW := avail(m, \sigma_n.res, \gamma.\delta, t);\\
& idBW := ideal(m, \sigma_n.res, \gamma.\delta, t) \ |\!)
\end{aligned}
$$

Proof. Similar to the proof of Lemma 5 □

Lemma 7.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = TRN(m, m', v, i, t) \;\Rightarrow\\
&\quad m \approx m' \;\wedge\; m'.ptr = m.ptr - 1 \;\wedge\; m'.accBW = newBW \;\#\; m.accBW \;\wedge\\
&\quad isBwd(v', i', m') \wedge\\
&\quad m \in \sigma_n.buf(v, i) \wedge m \notin \sigma_{n+1}.buf(v, i) \wedge m' \in \sigma_{n+1}.buf(v', i') \wedge\\
&\quad PathCheck(m') \;\wedge\; ResMsgCheck(m', \sigma_{n+1}.time) \wedge\\
&\quad \sigma_{n+1}.time = \sigma_n.time = t \;\wedge\\
&\quad atEnd(m) \;\wedge\; onPth(m') \wedge\\
&\quad \forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx.\\
&\qquad \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx) \wedge\\
&\quad ResMapCheck(\sigma_{n+1}.res, v, m) \;\wedge\; resCheck(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)
\end{aligned}$$

Proof. Similar to the proof of Lemma 5 □

Lemma 8.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = UPT(m, m', v, i, t) \;\Rightarrow\\
&\quad m \approx m' \;\wedge\; m'.ptr = m.ptr - 1 \;\wedge\; m'.accBW = m.accBW \;\wedge\\
&\quad isBwd(v', i', m') \wedge\\
&\quad m \in \sigma_n.buf(v, i) \wedge m \notin \sigma_{n+1}.buf(v, i) \wedge m' \in \sigma_{n+1}.buf(v', i') \wedge\\
&\quad PathCheck(m') \;\wedge\; ResMsgCheck(m', \sigma_{n+1}.time) \wedge\\
&\quad \sigma_{n+1}.time = \sigma_n.time = t \;\wedge\\
&\quad (onPth(m) \vee atSrt(m)) \wedge \big(onPth(m') \vee atSrt(m') \vee onWay(m')\big) \;\wedge\\
&\quad \forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx.\\
&\qquad \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx) \wedge\\
&\quad ResMapCheck(\sigma_{n+1}.res, v, m) \;\wedge\; resCheck(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)
\end{aligned}$$

Proof. Similar to the proof of Lemma 5 □

Lemma 9.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = BWD(m, m', v, i, t) \;\Rightarrow\\
&\quad\; m \approx m' \;\wedge\; m'.ptr = m.ptr - 1 \;\wedge\; m'.accBW = m.accBW \;\wedge\\
&\quad\; isBwd(v', i', m') \;\wedge\\
&\quad\; m \in \sigma_n.buf(v, i) \wedge m \notin \sigma_{n+1}.buf(v, i) \wedge m' \in \sigma_{n+1}.buf(v', i') \;\wedge\\
&\quad\; PathCheck(m') \;\wedge\; ResMsgCheck(m', \sigma_{n+1}.time) \;\wedge\\
&\quad\; \sigma_{n+1}.time = \sigma_n.time = t \;\wedge\\
&\quad\; onWay(m) \;\wedge\; onPth(m') \vee atSrc(m') \;\wedge\\
&\quad\; \forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx.\\
&\quad\;\quad \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx) \;\wedge\\
&\quad\; ResMapCheck(\sigma_{n+1}.res, v, m) \wedge resCheck(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)
\end{aligned}$$

Proof. Similar to the proof of Lemma 5 □

Lemma 10.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = FIN(m, m', v, i, t) \;\Rightarrow\\
&\quad\; m \approx m' \;\wedge\; m'.ptr = m.ptr - 1 \;\wedge\; m'.accBW = m.accBW \;\wedge\\
&\quad\; isBwd(v', i', m') \;\wedge\\
&\quad\; m \in \sigma_n.buf(v, i) \wedge m \notin \sigma_{n+1}.buf(v, i) \wedge m' \notin \sigma_{n+1}.buf(v', i') \;\wedge\\
&\quad\; PathCheck(m') \;\wedge\; ResMsgCheck(m', \sigma_{n+1}.time) \;\wedge\\
&\quad\; \sigma_{n+1}.time = \sigma_n.time = t \;\wedge\\
&\quad\; atSrc(m) \;\wedge\\
&\quad\; \forall \tilde{v} \neq v, \tilde{s} \neq src(m), \tilde{id} \neq m.id, \tilde{idx} \neq m.idx.\\
&\quad\;\quad \sigma_n.res(v, s, id).vrs(idx) = \sigma_{n+1}.res(v, s, id).vrs(idx) \;\wedge\\
&\quad\; ResMapCheck(\sigma_{n+1}.res, v, m) \;\wedge\; resCheck(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)
\end{aligned}$$

Proof. Similar to the proof of Lemma 5 □

B.2 Attacker Lemmata

Given a reservation message m containing only honest ASes on its path, i.e. $nodes(m) \subseteq H$. Then it holds that this message can neither be contained in the attacker knowledge nor in any buffer of an attacker. Note that for this proofs we use a slightly stronger version of the ATK event

$$
\begin{aligned}
& ATK(m \in \mathcal{M}, v \in V, i \in I) = \{(\sigma, \sigma') \mid \\
& \quad \texttt{- guards -} \\
& \quad m \in \sigma.kwl \land \\
& \quad \texttt{- actions -} \\
& \quad \sigma'.buf = \sigma.buf\,((v,i) \mapsto \sigma.buf(v,i) \cup \{m\})\ \}.
\end{aligned}
$$

Lemma 11.

$$
\begin{aligned}
& \forall m \in \mathcal{M}_R. \\
& \ nodes(m) \subseteq H \ \Rightarrow \\
& \quad \forall n \in \mathbb{N}, a \in M, i \in I.\ m \notin \sigma_n.kwl \ \land \ m \notin \sigma_n.buf\,(a,i).
\end{aligned}
$$

Proof. We show by induction on $n \in \mathbb{N}$

$$
\begin{aligned}
& \forall n \in \mathbb{N}, a \in M, i \in I, m \in \mathcal{M}_R. \\
& \ nodes(m) \subseteq H \ \Rightarrow \\
& \quad m \notin \sigma_n.kwl \ \land \ m \notin \sigma_n.buf\,(a,i)
\end{aligned}
$$

i.e. the induction hypothesis *IH* for σ_n is given by

$$
\begin{aligned}
& \forall a \in M, i \in I, m \in \mathcal{M}_R. \\
& (*)\ nodes(m) \subseteq H \ \Rightarrow \\
& \quad (1)\ m \notin \sigma_n.kwl \land \\
& \quad (2)\ m \notin \sigma_n.buf\,(a,i)
\end{aligned}
$$

$n = 0$: Since by definition of the initial state for any $m \in \sigma_0.kwl$ it holds that $src(m) \in A$ this leads to a contradictions with the assumption $nodes(m) \subseteq H$, i.e. (1) holds. By the definition of the initial state that $\forall v \in V, i \in I.\ \sigma_0.buf\,(v,i) = \emptyset$, the statement (2) holds trivially.

$n \rightarrow n+1$: By case distinction on λ_n,

$CLT(\hat{m}, \hat{m}', \hat{a}, \hat{i}, \hat{t})$: Hence, $\hat{m} \approx \hat{m}'$, $\hat{m} \in \sigma_n.buf\,(\hat{a}, \hat{i})$, and $\hat{a} \in M$

(1): Assume $m \in \sigma_{n+1}.kwl = \sigma_n.kwl \cup \{\hat{m}'\}$. By case distinction:

$m \in \sigma_n.kwl \setminus \{\hat{m}'\}$**:** Contradiction to *IH* (1).

$m = \hat{m}'$**:** By guard of CLT it holds that $\hat{m} \in \sigma_n.buf(\hat{a}, \hat{i})$ and $\hat{m} \approx \hat{m}'$. By $m = \hat{m}'$ and $(*)$ follows, that $nodes(\hat{m}) = nodes(m) \subseteq= H$, which is in contradiction to *IH* (2) applied to $\hat{m}$

(2): Assume $\exists a \in M, i \in I.\ m \in \sigma_{n+1}.buf(a,i)$. Due to the action of CLT it holds that $\sigma_{n+1}.buf(a,i) = \sigma_n.buf(a,i)$. This is in contradiction to *IH* (2) applied to m.

$RES(\hat{m}, \hat{m}', \hat{v}, \hat{i}, \hat{t})$**:** Hence,

$$(+)\ \sigma_{n+1}.buf = \sigma_n.buf\begin{pmatrix}(\hat{v},\hat{i}) & \mapsto & \sigma_n.buf(\hat{v},\hat{i}) \setminus \{\hat{m}\} \\ (\hat{v}',\hat{i}') & \mapsto & \sigma_n.buf(\hat{v}',\hat{i}') \cup \{\hat{m}'\}\end{pmatrix}$$

with $\hat{v}' = \hat{m}'.path[\hat{m}'.ptr].as$.

(1): Assume $m \in \sigma_{n+1}.kwl = \sigma_n.kwl$. By the action of RES, it holds that $m \in \sigma_{n+1}.kwl = \sigma_n.kwl$, which is in contradiction to *IH* (1) applied to m.

(2): Assume $\exists a \in M, i \in I.\ m \in \sigma_{n+1}.buf(a,i)$ By case distinction due to (+):

$m \in \sigma_n.buf(\hat{v}, \hat{i}) \setminus \{\hat{m}'\}$**:** Contradiction to *IH* (1).

$m = \hat{m}'$**:** By this it follows that $a = \hat{v}'$, hence, $a \in nodes(\hat{m}')$. Together with $\hat{m} \approx \hat{m}'$, hence $nodes(\hat{m}) = nodes(\hat{m}')$, and $(*)$ it follows, that

$$a = \hat{v}' \in nodes(\hat{m}) = nodes(m) \subseteq= H,$$

which is in contradiction to *IH* (2) applied to $\hat{m}$

$CRT_R(\hat{m}, \hat{v}, \hat{i}, \hat{t})$**:** Due to the action of CRT_R, it holds

$$(+)\ \sigma_{n+1}.buf = \sigma_n.buf\left((\hat{v},\hat{i}) \mapsto \sigma_n.buf(\hat{v},\hat{i}) \cup \{\hat{m}\}\right)$$

(1): Assume $m \in \sigma_{n+1}.kwl$. By the action of CRT_R, it holds that $m \in \sigma_{n+1}.kwl = \sigma_n.kwl$, which is in contradiction to *IH* (1) applied to m.

(2): Assume $\exists a \in M, i \in I.\ m \in \sigma_{n+1}.buf(a,i)$. By case distinction on (+) it follows:

$m \in \sigma_n.buf(\hat{v},\hat{i}) \setminus \{\hat{m}\}$: By assumption $(\hat{v},\hat{i}) = (a,i)$, which contradicts *IH* (2) for m.

$m = \hat{m}$: By this it follows that $src(m) = src(\hat{m})$. By the guards of CRT_R it follows that $src(\hat{m}) = \hat{v}$ and by (+) and assumption it follows that $\hat{v} = a$, hence,

$$a = src(\hat{m}) = src(m) \in nodes(m) \subseteq H,$$

i.e. a contradiction to $(*)$ for m.

$ATK(\hat{m},\hat{v},\hat{i},\hat{t})$: Due to the action of ATK, it holds

$$(+)\ \sigma_{n+1}.buf = \sigma_n.buf\left((\hat{v},\hat{i}) \mapsto \sigma_n.buf(\hat{v},\hat{i}) \cup \{\hat{m}\}\right)$$

(1): Assume $m \in \sigma_{n+1}.kwl$. By the action of ATK, it holds that $m \in \sigma_{n+1}.kwl = \sigma_n.kwl$, which contradicts *IH* (1) applied to m.

(2): Assume $\exists a \in M, i \in I.\ m \in \sigma_{n+1}.buf(a,i)$. By case distinction on (+) it follows:

$m \in \sigma_n.buf(\hat{v},\hat{i}) \setminus \{\hat{m}\}$: By assumption $(\hat{v},\hat{i}) = (a,i)$, which contradicts *IH* (2) for m.

$m = \hat{m}$: By this and the guard of ATK it follows that $m = \hat{m} \in \sigma_n.kwl$, which is in contradiction to *IH* (1) for m.

other : Both fields buf and kwl stay unchanged.

□

B.3 In Buffer Lemmata

Honest

Lemma 12.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&isBwd(v,e,m)\ \wedge\ 0 \le m.ptr < m.first - 1\ \wedge\\
&nodes(m) \subseteq H\ \wedge\ m \in \sigma_n.buf(v,e) \Rightarrow\\
&\ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = BWD(\bar{v},\bar{e},\bar{m},\bar{m}',\bar{t})\ \wedge\\
&\quad m = \bar{m}'\ \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \le n \Rightarrow m \in \sigma_{\hat{n}}.buf(v,e)
\end{aligned}$$

Proof. By induction on n:

$n = 0$: The premise $\lambda_0 = FIN(m, m', v, e, t)$ cannot hold, since $\sigma_0.buf = \emptyset$, hence the event's guard $m \in \sigma_0.buf(v, e)$ is not satisfied and the claim holds trivially.

$n \to n+1$: Assume *IH*:

$$\begin{aligned}
&\forall m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&isBwd(v, e, m) \wedge 0 \leq m.ptr < m.first - 1 \wedge\\
&nodes(m) \subseteq H \wedge m \in \sigma_n.buf(v, e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad\quad \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\quad\quad m = \bar{m}' \wedge\\
&\quad\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Given $m, m' \in \mathcal{M}_R$, $v \in H$, $e \in I$, $t \in \mathbb{N}$ with

$$\begin{aligned}
&(1)\ isBwd(v, e, m) \wedge\\
&(2)\ 0 \leq m.ptr < m.first - 1 \wedge\\
&(3)\ nodes(m) \subseteq H \wedge\\
&(4)\ m \in \sigma_{n+1}.buf(v, e)
\end{aligned}$$

We need to show:

$$\begin{aligned}
&\exists \bar{n} < n+1, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}. \bar{n} < \hat{n} \leq n+1.\ m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

By case distinction on λ_n:

$TCK(\tilde{t})$: The event's actions keep the buffer the same, i.e., $m \in \sigma_{n+1}.buf(v, e)$ implies $m \in \sigma_n.buf(v, e)$. Given (1),(2) and (3) we can apply *IH* and obtain:

$$\begin{aligned}
&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Since $n < n+1$ we only need to show $m \in \sigma_{n+1}.\mathit{buf}(v,e)$ in the fourth line, which is given by (4).

$RST(\tilde{v}, \tilde{s}, \tilde{id}, \tilde{idx})$: As in TCK.

$CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: Two cases:

$m = \tilde{m} \wedge v = \tilde{v} \wedge e = \tilde{e}$: By Lemma 4 it follows $\mathit{isFwd}(v,e,m)$ in contradiction to (1).

Otherwise: As in TCK.

$CRT_D(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: As CRT_R, but in the case $\tilde{m} = m$ the contradiction comes from both messages having different types.

$FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: Following cases:

$m = \tilde{m}$: By Lemma 5 it follows $\mathit{isFwd}(v,e,m)$ in contradiction to (1).[1]

$m = \tilde{m}'$: By Lemma 5 it follows $\mathit{isFwd}(v,e,m)$ in contradiction to (1).

Otherwise: As in TCK.

$CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As FWD.

$TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As FWD.

$UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As FWD, but the contradiction comes from both messages $\tilde{m}$ and $\tilde{m}'$ have the following locations, i.e.,

$$\tilde{m}.\mathit{first} \le \tilde{m}.ptr \le \tilde{m}.\mathit{last} - 1$$
$$\tilde{m}.\mathit{first} - 1 \le \tilde{m}'.ptr \le \tilde{m}.\mathit{last} - 2$$

Hence in case $m = \tilde{m} \vee m = \tilde{m}'$ this contradicts assumption (2).

$BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: Following cases:

$m = \tilde{m}'$: Set $\bar{n} := n$ (hence $\bar{n} < n+1$). Hence by setting $\bar{m} := \tilde{m}$, $\bar{m}' := \tilde{m}'$ (hence $m = \tilde{m}' = \bar{m}'$), $\bar{v} := \tilde{v}$, $\bar{e} := \tilde{e}$, and $\bar{t} := \tilde{t}$ it follows:

$$\begin{aligned}
&\exists \bar{n} < n+1, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\quad m = \bar{m}'
\end{aligned}$$

Since $n =: \bar{n} < \hat{n} \le n+1$, i.e. $\hat{n} = n+1$, and $m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)$ by (4) it follows trivially:

$$\forall \hat{n}.\ \bar{n} < \hat{n} \le n+1 \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)$$

[1] $m \notin \sigma_{n+1}.\mathit{buf}(\tilde{v}, \tilde{e})$ in contradiction to (4).

$m = \tilde{m}$: By Lemma 5 it follows

$$\forall \tilde{v} \in V, \tilde{e} \in I.\ m \notin \sigma_{n+1}.buf(\tilde{v}, \tilde{e})$$

in contradiction to (4).

$m \neq \tilde{m} \vee m \neq \tilde{m}'$: As in TCK.

$FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As CRT_R, but in the case $\tilde{m} = m$ the contradiction comes from $m \notin \sigma_{n+1}.buf(v, e)$ and (4).

$CLT(\tilde{m}, \tilde{m}', \tilde{a}, \tilde{i}, \tilde{t})$: As in TCK.

$ATK(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: Two cases:

$m \approx \tilde{m}$: By assumption $nodes(m) =^{m \approx \tilde{m}} nodes(\tilde{m}) \subseteq H$ and by Lemma 11 it follows that $\tilde{m} \notin \sigma_n.kwl$ in contradiction to the event's guard.

$m \not\approx \tilde{m}$: This implies $m \neq \tilde{m}$, hence as in TCK.

$RMV(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As in FIN.

$DST(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As in FIN.

$DRP(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: As in FIN.

□

Lemma 13.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\quad isBwd(v, e, m)\ \wedge\ m.first - 1 \leq m.ptr < m.last - 1\ \wedge\\
&\quad nodes(m) \subseteq H\ \wedge\ m \in \sigma_n.buf(v, e) \Rightarrow\\
&\quad\ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad\quad \lambda_{\bar{n}} = UPT(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t})\ \wedge\\
&\quad\quad m = \bar{m}'\ \wedge\\
&\quad\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 12 □

Lemma 14.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isBwd}(v,e,m) \ \wedge\ m.ptr = m.last - 1 \wedge\\
&\ \mathit{nodes}(m) \subseteq H \ \wedge\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\qquad \lambda_{\bar{n}} = TRN(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\qquad m = \bar{m}' \wedge\\
&\qquad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 12 □

Lemma 15.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isFwd}(v,e,m) \ \wedge\ m.\mathit{first} < m.ptr \leq m.last \wedge\\
&\ \mathit{nodes}(m) \subseteq H \ \wedge\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\qquad \lambda_{\bar{n}} = CMP(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\qquad m = \bar{m}' \wedge\\
&\qquad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 12 □

Lemma 16.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isFwd}(v,e,m) \ \wedge\ 0 < m.ptr \leq m.\mathit{first} \wedge\\
&\ \mathit{nodes}(m) \subseteq H \ \wedge\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\qquad \lambda_{\bar{n}} = FWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\qquad m = \bar{m}' \wedge\\
&\qquad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 12 □

Lemma 17.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\mathit{isFwd}(v,e,m) \ \wedge \ m.ptr = 0 \wedge\\
&\mathit{nodes}(m) \subseteq H \ \wedge \ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\ \exists \bar{n} < n, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = CRT_R(\bar{v}, \bar{e}, \bar{m}', \bar{t}) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 12 □

General

Lemma 18.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\mathit{isBwd}(v,e,m) \ \wedge \ 0 \leq m.ptr < m.\mathit{first} - 1 \wedge\\
&m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ (\ \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \vee\\
&\quad \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \wedge\\
&\ m = \bar{m}' \wedge\\
&\ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. As in Lemma 12, except in case:
$\lambda_n = ATK(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: Two cases:

$m = \tilde{m}$: Set $\bar{n} := n$, i.e., $\bar{n} < n+1$, and $\bar{m'} := \tilde{m} = m$, $\bar{v} := \tilde{v}$, $\bar{e} := \tilde{i}$, $\bar{t} := \tilde{t}$, hence,

$$\lambda_{\bar{n}} = ATK(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t}) = ATK(\bar{m}, \bar{v}, \bar{i}, \bar{t})$$

By assumption $m \in \sigma_{n+1}.\mathit{buf}(v,e)$ the claim follows.

$m \neq \tilde{m}$: As in TCK.

□

Lemma 19.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isBwd}(v,e,m) \ \wedge\ m.\mathit{first} - 1 \leq m.ptr < m.\mathit{last} - 1 \ \wedge\\
&\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad (\ \lambda_{\bar{n}} = UPT(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \vee\\
&\quad\quad \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 18 □

Lemma 20.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isBwd}(v,e,m) \ \wedge\ m.ptr = m.\mathit{last} - 1 \ \wedge\\
&\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad (\ \lambda_{\bar{n}} = TRN(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \vee\\
&\quad\quad \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 18 □

Lemma 21.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isFwd}(v,e,m) \ \wedge\ m.\mathit{first} < m.ptr \leq m.\mathit{last} \ \wedge\\
&\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad (\ \lambda_{\bar{n}} = CMP(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \vee\\
&\quad\quad \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 18 □

Lemma 22.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isFwd}(v,e,m)\ \wedge\ 0 < m.ptr \le m.\mathit{first} \wedge\\
&\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad\ (\ \lambda_{\bar{n}} = FWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \vee\\
&\quad\quad \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \wedge\\
&\quad\ m = \bar{m}' \wedge\\
&\quad\ \forall \hat{n}.\ \bar{n} < \hat{n} \le n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 18 □

Lemma 23.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isFwd}(v,e,m)\ \wedge\ m.ptr = 0 \wedge\\
&\ m \in \sigma_n.\mathit{buf}(v,e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad\ \sigma_n - \sigma_{\bar{n}} \le \mathit{bufT} \wedge\\
&\quad\ (\ \lambda_{\bar{n}} = CRT_R(\bar{v}, \bar{e}, \bar{m}', \bar{t}) \vee\\
&\quad\quad \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \wedge\\
&\quad\ m = \bar{m}' \wedge\\
&\quad\ \forall \hat{n}.\ \bar{n} < \hat{n} \le n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v,e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 18 □

Lemma 24.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \mathit{isWrg}(v, e, m) \ \lor \ m.ptr > m.\mathit{last} \land\\
&\ m \in \sigma_n.\mathit{buf}(v, e) \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad\ \sigma_n - \sigma_{\bar{n}} \leq \mathit{bufT} \land\\
&\quad\ \lambda_{\bar{n}} = ATK(\bar{v}, \bar{e}, \bar{m}', \bar{t})\ ) \land\\
&\quad\ m = \bar{m}' \land\\
&\quad\ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v, e)
\end{aligned}$$

Proof. Similar to the proof of Lemma 18 □

B.4 Step Lemmata

Lemma 25.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = FIN(m, m', v, e, t) \land \mathit{nodes}(m) \subseteq H \Rightarrow\\
&\quad \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad\ \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \land\\
&\quad\ m = \bar{m}' \land\\
&\quad\ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v, e)
\end{aligned}$$

Proof. Given $m, m' \in \mathcal{M}_R$, $v \in H$, $e \in I$, $t \in \mathbb{N}$ with $k = m.ptr$, $\mathit{nodes}(m) \subseteq H$ and $\lambda_n = FIN(m, m', v, e, t)$.

By Lemma 10 it follows that $m \in \sigma_n.\mathit{buf}(v, e)$ with $\mathit{isBwd}(v, e, m)$ and $\mathit{atSrc}(m)$.

By Lemma 18 we get:

$$\begin{aligned}
&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \land\\
&\ m = \bar{m}' \land\\
&\ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.\mathit{buf}(v, e)
\end{aligned}$$

□

Lemma 26.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = BWD(m, m', v, e, t) \land nodes(m) \subseteq H \land\\
&\ \ 0 \le m.ptr < m.first \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = BWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \land\\
&\ \ \ m = \bar{m}' \land\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \le n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 27.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = BWD(m, m', v, e, t) \land nodes(m) \subseteq H \land\\
&\ \ m.ptr = m.first - 1 \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = UPT(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \land\\
&\ \ \ m = \bar{m}' \land\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \le n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 28.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = UPT(m, m', v, e, t) \land nodes(m) \subseteq H \land\\
&\ \ m.first \le m.ptr < m.last - 1 \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = UPT(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \land\\
&\ \ \ m = \bar{m}' \land\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \le n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 29.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = UPT(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\
&\ \ m.ptr = m.last - 1 \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = TRN(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\ \ \ m = \bar{m}' \wedge\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 30.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = TRN(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\
&\ \ m.ptr = m.last \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = CMP(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\ \ \ m = \bar{m}' \wedge\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 31.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = CMP(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\
&\ \ m.first < m.ptr < m.last \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = CMP(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\ \ \ m = \bar{m}' \wedge\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 32.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = CMP(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\
&\ \ m.ptr = m.\mathit{first} \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = FWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\ \ \ m = \bar{m}' \wedge\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 33.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = FWD(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\
&\ \ 0 < m.ptr < m.\mathit{first} \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = FWD(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\ \ \ m = \bar{m}' \wedge\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

Lemma 34.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\
&\ \lambda_n = FWD(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\
&\ \ m.ptr = 0 \Rightarrow\\
&\ \ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\ \ \ \lambda_{\bar{n}} = CRT_R(\bar{m}, \bar{v}, \bar{e}, \bar{t}) \wedge\\
&\ \ \ m = \bar{m}' \wedge\\
&\ \ \ \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Proof. Similarly to the proof in Lemma 25. □

B.5 Uniqueness Lemmata

Lemma 35.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m_1', m_2, m_2' \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = FIN(m_1, m_1', v_1, e_1, t_1) \land\\
&\lambda_{n_2} = FIN(m_2, m_2', v_2, e_2, t_2) \land\\
&m_1 \sim m_2 \ \land \ v_1 = v_2 \land\\
&nodes(m_1) \subseteq H \ \land\\
&n_1 < n_2 \ \land \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Proof. By $m_1 \sim m_2$, $m_1.ptr = m_2.ptr$, and $m_1.path = m_2.path$ it follows that $v := v_1 = v_2$ and $e := e_1 = e_2$. Together with assumption $nodes(m_1) \subseteq H$ by Lemma 25 it follows that

$$\begin{aligned}
(1)\ \ &\exists \bar{n}_1 < n_1, \bar{m}_1, \bar{m}_1' \in \mathcal{M}_R, \bar{v}_1 \in H, \bar{e}_1 \in I, \bar{t}_1 \in \mathbb{N}.\\
&\lambda_{\bar{n}_1} = BWD(\bar{m}_1, \bar{m}_1', \bar{v}_1, \bar{e}_1, \bar{t}_1) \land\\
&m_1 = \bar{m}_1' \ \land\\
&\forall \hat{n}_1.\ \bar{n}_1 < \hat{n}_1 \leq n_1 \Rightarrow m_1 \in \sigma_{\hat{n}_1}.buf(v, e)
\end{aligned}$$

and similarly with $m_1.path = m_2.path$ and $nodes(m_2) \subseteq H$ follows (2) for n_2.

$$\begin{aligned}
(2)\ \ &\exists \bar{n}_2 < n_2, \bar{m}_2, \bar{m}_2' \in \mathcal{M}_R, \bar{v}_2 \in H, \bar{e}_2 \in I, \bar{t}_2 \in \mathbb{N}.\\
&\lambda_{\bar{n}_2} = BWD(\bar{m}_2, \bar{m}_2', \bar{v}_2, \bar{e}_2, \bar{t}_2) \land\\
&m_2 = \bar{m}_2' \ \land\\
&\forall \hat{n}_2.\ \bar{n}_2 < \hat{n}_2 \leq n_2 \Rightarrow m_2 \in \sigma_{\hat{n}_2}.buf(v, e)
\end{aligned}$$

By $m_1 \sim m_2$ and $m_1 = \bar{m}_1'$ and $\bar{m}_1' \approx \bar{m}_1$ (and analogously for $\bar{m}_2$), it follows that $\bar{m}_1 \sim \bar{m}_2$ Furthermore it follows:

$$\begin{aligned}
nodes(\bar{m}_1) &=^{BWD(\bar{m}_1, \bar{m}_1', \dots)} nodes(\bar{m}_1')\\
&=^{m_1 = \bar{m}_1'} nodes(m_1) \subseteq H
\end{aligned}$$

and

$$\begin{aligned}\bar{m}_1.ptr &=^{BWD(\bar{m}_1,\bar{m}_1',\dots)} \bar{m}_1'.ptr-1\\ &=^{m_1=\bar{m}_1'} m_1.ptr-1\\ &=^{m_1.ptr=m_2.ptr} m_2.ptr-1\\ &=^{m_2=\bar{m}_2'} \bar{m}_2'.ptr-1 =^{BWD(\bar{m}_2,\bar{m}_2',\dots)} \bar{m}_2.ptr\end{aligned}$$

and

$$\begin{aligned}\bar{m}_1.path &=^{BWD(\bar{m}_1,\bar{m}_1',\dots)} \bar{m}_1'.path\\ &=^{m_1=\bar{m}_1'} m_1.path\\ &=^{m_1.path=m_2.path} m_2.path\\ &=^{m_2=\bar{m}_2'} \bar{m}_2'.path =^{BWD(\bar{m}_2,\bar{m}_2',\dots)} \bar{m}_2.path\end{aligned}$$

Three cases:

$\bar{n}_1 < \bar{n}_2$ Then it holds:

$$\begin{aligned}\sigma_{\bar{n}_2}.time &\leq^{\bar{n}_2<n_2} \sigma_{n_2}.time\\ &\leq^{\sigma_{n_2}.time\leq m_1.expT} m_1.expT\\ &=^{m_1=\bar{m}_1'} \bar{m}_1'.expT\\ &=^{BWD(\bar{m}_1,\bar{m}_1',\dots)} \bar{m}_1.expT\end{aligned}$$

By Lemma 36 applied to

$$\begin{aligned}&\lambda_{\bar{n}_1} = BWD(\bar{m}_1,\bar{m}_1',\bar{v}_1,\bar{e}_1,\bar{t}_1)\wedge\\ &\lambda_{\bar{n}_2} = BWD(\bar{m}_2,\bar{m}_2',\bar{v}_2,\bar{e}_2,\bar{t}_2)\wedge\\ &\bar{m}_1 \sim \bar{m}_2 \ \wedge\ \bar{m}_1.ptr=\bar{m}_2.ptr \ \wedge\ \bar{m}_1.path=\bar{m}_2.path \wedge\\ &nodes(\bar{m}_1)\subseteq H \wedge\\ &\bar{n}_1<\bar{n}_2 \ \wedge\ \sigma_{\bar{n}_2}.time \leq \bar{m}_1.expT\end{aligned}$$

it follows $\bar{n} := \bar{n}_1 = \bar{n}_2$.

$\bar{n}_2 < \bar{n}_1$: Then it holds:

$$\begin{aligned}\sigma_{\bar{n}_1}.time &\leq^{\bar{n}_1 < n_1} \sigma_{n_1}.time \\ &\leq^{n_1 < n_2} \sigma_{n_2}.time \\ &\leq^{\sigma_{n_2}.time \leq m_2.expT} m_2.expT \\ &=^{m_2 = \bar{m}'_2} \bar{m}'_2.expT \\ &=^{BWD(\bar{m}_2, \bar{m}'_2, \dots)} \bar{m}_2.expT\end{aligned}$$

By Lemma 36 applied to

$$\begin{aligned}&\lambda_{\bar{n}_1} = BWD(\bar{m}_1, \bar{m}'_1, \bar{v}_1, \bar{e}_1, \bar{t}_1) \wedge \\ &\lambda_{\bar{n}_2} = BWD(\bar{m}_2, \bar{m}'_2, \bar{v}_2, \bar{e}_2, \bar{t}_2) \wedge \\ &\bar{m}_1 \sim \bar{m}_2 \ \wedge \ \bar{m}_1.ptr = \bar{m}_2.ptr \ \wedge \ \bar{m}_1.path = \bar{m}_2.path \wedge \\ &nodes(\bar{m}_2) \subseteq H \ \wedge \\ &\bar{n}_2 < \bar{n}_1 \ \wedge \ \sigma_{\bar{n}_1}.time \leq \bar{m}_2.expT\end{aligned}$$

it follows $\bar{n} := \bar{n}_1 = \bar{n}_2$.

$\bar{n}_1 = \bar{n}_2$: Hence, $\bar{n} := \bar{n}_1 = \bar{n}_2$ holds immediately.

Since π is a function and $\lambda_{\bar{n}_1} = \lambda_{\bar{n}_2}$ and therefore $\bar{m} := \bar{m}_1 = \bar{m}_2$, $\bar{m}' := \bar{m}'_1 = \bar{m}'_2$, $\bar{v} := \bar{v}_1 = \bar{v}_2$, $\bar{e} := \bar{e}_1 = \bar{e}_2$, and $\bar{t} := \bar{t}_1 = \bar{t}_2$. Set $m := m_1 = \bar{m}'_1 = \bar{m}'_2 = m_2$, $\bar{n} := \bar{n}_1 = \bar{n}_2$ in (2), then we get:

$$(2') \quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n_2 \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)$$

By assumption $\lambda_{n_1} = FIN(m_1, m'_1, v_1, e_1, t_1)$ and $m := m_1$ we get $m \notin \sigma_{n_1+1}.buf(v, e)$.

By $n_1 < n_2$ we get $\tilde{n} < n_1 + 1 \leq n_2$. By setting $\hat{n} := n_1 + 1$ in $(2')$ we get $m \in \sigma_{n_1+1}.buf(v, e)$, i.e. a contradiction. □

Corollary 6.

$$\begin{aligned}&\forall n_1, n_2 \in \mathbb{N}, m_1, m'_1, m_2, m'_2 \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}. \\ &\lambda_{n_1} = FIN(m_1, m'_1, v_1, e_1, t_1) \wedge \\ &\lambda_{n_2} = FIN(m_2, m'_2, v_2, e_2, t_2) \wedge \\ &m_1 \approx m_2 \wedge m_1.ptr = m_2.ptr \ \wedge \ nodes(m_1) \subseteq H \\ &\Rightarrow n_1 = n_2\end{aligned}$$

Lemma 36.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m_1', m_2, m_2' \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = BWD(m_1, m_1', v_1, e_1, t_1) \land\\
&\lambda_{n_2} = BWD(m_2, m_2', v_2, e_2, t_2) \land\\
&m_1 \sim m_2 \ \land \ v_1 = v_2 \land\\
&nodes(m_1) \subseteq H \land\\
&n_1 < n_2 \ \land \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Proof. Similar to the proof of Lemma 35. □

Lemma 37.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m_1', m_2, m_2' \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = UPT(m_1, m_1', v_1, e_1, t_1) \land\\
&\lambda_{n_2} = UPT(m_2, m_2', v_2, e_2, t_2) \land\\
&m_1 \sim m_2 \ \land \ v_1 = v_2 \land\\
&nodes(m_1) \subseteq H \land\\
&n_1 < n_2 \ \land \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Proof. Similar to the proof of Lemma 35. □

Lemma 38.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m_1', m_2, m_2' \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = TRN(m_1, m_1', v_1, e_1, t_1) \land\\
&\lambda_{n_2} = TRN(m_2, m_2', v_2, e_2, t_2) \land\\
&m_1 \sim m_2 \ \land \ v_1 = v_2 \land\\
&nodes(m_1) \subseteq H \land\\
&n_1 < n_2 \ \land \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Proof. Similar to the proof of Lemma 35. □

Lemma 39.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m'_1, m_2, m'_2 \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = CMP(m_1, m'_1, v_1, e_1, t_1) \wedge\\
&\lambda_{n_2} = CMP(m_2, m'_2, v_2, e_2, t_2) \wedge\\
&m_1 \sim m_2 \ \wedge \ v_1 = v_2 \wedge\\
&nodes(m_1) \subseteq H \wedge\\
&n_1 < n_2 \ \wedge \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Proof. Similar to the proof of Lemma 35. □

Lemma 40.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m'_1, m_2, m'_2 \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = FWD(m_1, m'_1, v_1, e_1, t_1) \wedge\\
&\lambda_{n_2} = FWD(m_2, m'_2, v_2, e_2, t_2) \wedge\\
&m_1 \sim m_2 \ \wedge \ v_1 = v_2 \wedge\\
&nodes(m_1) \subseteq H \wedge\\
&n_1 < n_2 \ \wedge \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Proof. Similar to the proof of Lemma 35. □

Lemma 41.

$$\begin{aligned}
&\forall n_1, n_2 \in \mathbb{N}, m_1, m_2 \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}.\\
&\lambda_{n_1} = CRT_R(m_1, v_1, e_1, t_1) \wedge\\
&\lambda_{n_2} = CRT_R(m_2, v_2, e_2, t_2) \wedge\\
&m_1 \sim m_2 \wedge\\
&n_1 < n_2 \ \wedge \ \sigma_{n_2}.time \leq m_1.expT\\
&\ \Rightarrow n_1 = n_2
\end{aligned}$$

Note that $m_1.ptr = m_2.ptr$ follows from the first two assumptions.

Furthermore, we do not need the assumption $nodes(m_1) \subseteq H$, but in the following lemmas we need it to use the respective existence lemmas.

Proof. By $m_1 \sim m_2$ it follows $src(m_1) = src(m_2)$. By $atSrc(m_1)$ and $isFwd(m_1, v_1, i_1)$ it follows $v_1 = src(m_1)$ and similarly $v_2 = src(m_2)$, hence, all together $v := v_1 = v_2$. Together with Lemma 44 applied to $\lambda_{n_1} = CRT_R(m_1, v, e_1, t_1)$ it follows that

$$\begin{aligned}(*)\ \forall \hat{n} > n_1.\ &\sigma_{\hat{n}}.time \leq m_1.expT \Rightarrow \\ &(\ isMrkd(\sigma_{\hat{n}}.res, v, m_1, \sigma_{\hat{n}}.time) \lor \\ &\ \ isRsvd(\sigma_{\hat{n}}.res, v, m_1, \sigma_{\hat{n}}.time)\ )\end{aligned}$$

Set $\hat{n} := n_2$ (by assumption $n_2 > n_1$).

By assumption $\sigma_{n_2}.time \leq m_1.expT$ and by $(*)$ then

$$\begin{aligned}(\ &isMrkd(\sigma_{n_2}.res, v, m_1, \sigma_{n_2}.time) \lor \\ &isRsvd(\sigma_{n_2}.res, v, m_1, \sigma_{n_2}.time)\ )\end{aligned}$$

Since $m_1 \sim m_2$ it follows by Lemma 43

$$\begin{aligned}(\ &isMrkd(\sigma_{n_2}.res, v, m_2, \sigma_{n_2}.time) \lor \\ &isRsvd(\sigma_{n_2}.res, v, m_2, \sigma_{n_2}.time)\ )\end{aligned}$$

However, this is in contradiction to the guard of $\lambda_{n_2} = CRT_R(m_2, v, e_2, t_2)$, i.e.

$$notRsvd(\sigma_{n_2}.res, v, m_2, \sigma_{n_2}.time)$$

, hence $n_1 = n_2$. □

B.6 Reserved Lemmata

Lemma 42.

$$\begin{aligned}&\forall res \in ResMap, v \in V m, m' \in \mathcal{M}, t, t' \in \mathbb{N}. \\ &\ m \approx m' \ \land\ m.ptr = m'.ptr \ \land\ ResMapCheck(res, v, m) \\ &\ \Rightarrow ResMapCheck(res, v, m')\end{aligned}$$

Proof. By definition of *ResMapCheck*

$$
\begin{aligned}
&(0)\ res(v,src(m),m.id) = \bot \lor\\
&(\ \neg(0)\ res(v,src(m),m.id) \neq \bot \land\\
&\quad (1)\ res(v,src(m),m.id).path = m.path \land\\
&\quad (2)\ res(v,src(m),m.id).ptr = m.ptr \land\\
&\quad (3)\ res(v,src(m),m.id).first = m.first \land\\
&\quad (4)\ res(v,src(m),m.id).last = m.last \land\\
&\quad (\ (5)\ res(v,src(m),m.id).vrs = \emptyset \lor\\
&\qquad \neg(5)\ res(v,src(m),m.id).vrs \neq \emptyset \land\\
&\qquad \neg(6)\ res(v,src(m),m.id).vrs(m.idx) \neq \bot \land\\
&\qquad (7)\ res(v,src(m),m.id).vrs(m.idx).minBW = m.minBW \land\\
&\qquad (8)\ res(v,src(m),m.id).vrs(m.idx).maxBW = m.maxBW \land\\
&\qquad (9)\ res(v,src(m),m.id).vrs(m.idx).expT = m.expT\ )\ )
\end{aligned}
$$

and by $m \approx m'$ implying

$$
\begin{aligned}
&src(m) = src(m') \land\\
&m.id = m'.id \land\\
&m.idx = m'.idx \land\\
&m.path = m'.path \land\\
&m.first = m'.first \land\\
&m.last = m'.last \land\\
&m.minBW = m'.minBW \land\\
&m.maxBW = m'.maxBW \land\\
&m.expT = m'.expT
\end{aligned}
$$

and $m.ptr = m'.ptr$ it follows *ResMapCheck*(res, v, m') by replacing the corresponding fields. □

Lemma 43.

$$
\begin{aligned}
&\forall res \in ResMap, v \in V, m, m' \in \mathcal{M}_R, t, t' \in \mathbb{N}.\\
&\quad m \sim m' \ \land\ t \geq t' \ \land\ isMrkd(res, v, m, t)\\
&\qquad \Rightarrow isMrkd(res, v, m', t')
\end{aligned}
$$

Same holds true for *isRsvd* instead of *isMrkd*.

Proof. Assume $\mathit{isMrkd}(res, v, m, t)$, i.e.

$$\begin{aligned}
&\neg(0)\ res(v, src(m), m.id) \neq \bot\ \wedge \\
&\neg(5)\ res(v, src(m), m.id).vrs \neq \emptyset\ \wedge \\
&\neg(6)\ res(v, src(m), m.id).vrs(m.idx) \neq \bot\ \wedge \\
&(X)\ res(v, src(m), m.id).vrs(m.idx).resBW = \bot\ \wedge \\
&(9')\ res(v, src(m), m.id).vrs(m.idx).expT \geq t
\end{aligned}$$

By $m \sim m'$, i.e.

$$src(m) = src(m')\ \wedge\ m.id = m'.id\ \wedge\ m.idx = m'.idx$$

it follows $\mathit{isMrkd}(res, v, m', t)$ and by $t \geq t'$ it follows $(9')$ and therefore $\mathit{isMrkd}(res, v, m', t')$. □

Lemma 44 (CRT-isMrkd-until-expiration).

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}. \\
&\ \lambda_n = CRT(m, v, i, t)\ \Rightarrow \\
&\ \ \forall \hat{n} > n.\ \sigma_{\hat{n}}.time \leq m.expT \Rightarrow \\
&\ \ \ (\ \mathit{isMrkd}(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\
&\ \ \ \ \ \vee \mathit{isRsvd}(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ )\ \wedge \\
&\ \ \ \mathit{ResMapCheck}(\sigma_{\hat{n}}.res, v, m)
\end{aligned}$$

Proof. Given $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$ with m, $v \in V$, $i \in I$, $t \in \mathbb{N}$ with

$$\lambda_n = CRT(m, v, i, t)$$

By induction on $\hat{n}$:

$\hat{n} = n + 1$: By assumption $\lambda_n = CRT(m, v, i, t)$ and Lemma 4 it holds $\mathit{isMrkd}(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)$ and $\mathit{ResMapCheck}(\sigma_{n+1}.res, v, m)$, hence, in this case $(\hat{n} = n + 1)$ the claim.

$\hat{n} \rightarrow \hat{n} + 1$: By *IH* it holds

$$\begin{aligned}
&\hat{n} > n\ \wedge\ \sigma_{\hat{n}}.time \leq m.expT \Rightarrow \\
&(\ \mathit{isMrkd}(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\
&\ \ \vee \mathit{isRsvd}(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ )\ \wedge \\
&\mathit{ResMapCheck}(\sigma_{\hat{n}}.res, v, m)
\end{aligned}$$

We need to show:

$$\begin{aligned}&\hat{n}+1 > n \;\wedge\; \sigma_{\hat{n}+1}.time \leq m.expT \Rightarrow\\&(\; isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)\\&\quad \vee isRsvd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)\;) \wedge\\&ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)\end{aligned}$$

Assume $\hat{n}+1 > n+1$ and $\sigma_{\hat{n}+1}.time \leq m.expT$.

Case distinction by $\lambda_{\hat{n}}$:

$TCK(\tilde{t})$: By the event's guard it holds $\sigma_{\hat{n}}.time = \tilde{t}$. Two cases:

$\sigma_{\hat{n}}.time \geq m.expT$: In this case

$$\sigma_{\hat{n}+1}.time = \sigma_{\hat{n}}.time + 1 > m.expT$$

hence, the premise is not satisfied, i.e. the claim is true.

$\sigma_{\hat{n}}.time < m.expT$: In this case it immediately holds $\sigma_{\hat{n}}.time \leq m.expT$ and we can apply IH and get:

$$\begin{aligned}&(\; isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\\&\quad \vee isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\;) \wedge\\&ResMapCheck(\sigma_{\hat{n}}.res, v, m)\end{aligned}$$

Since the event's action does not change the reservation maps, i.e. $\sigma_{\hat{n}}.res = \sigma_{\hat{n}+1}.res$, it follows that

$$\begin{aligned}&(\; isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}}.time)\\&\quad \vee isRsvd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}}.time)\;) \wedge\\&ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)\end{aligned}$$

By (9) of $ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)$ and assumption $\sigma_{\hat{n}+1}.time \leq m.expT$ it follows that

$$\begin{aligned}(9')\; &\sigma_{\hat{n}+1}(v, src(m), m.id).vrs(m.idx).expT\\&=^{(9)} m.expT \geq \sigma_{\hat{n}+1}.time\end{aligned}$$

i.e. $isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}}.time)$ and therefore the claim.

$RST(\tilde{v}, \tilde{s}, \tilde{id}, \tilde{idx})$: The event's actions do not change time, i.e. $\sigma_{\hat{n}+1}.time = \sigma_{\hat{n}}.time$. Together with the assumptions $\hat{n}+1 > n+1$ and $\sigma_{\hat{n}+1}.time \leq m.expT$ we get

$$\hat{n} > n$$
$$\sigma_{\hat{n}}.time \leq m.expT$$

and can apply *IH* and get:

$$\begin{aligned}(-)\ \ (\ & isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\ & \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \land \\ & ResMapCheck(\sigma_{\hat{n}}.res, v, m)\end{aligned}$$

and applying $\sigma_{\hat{n}+1}.time = \sigma_{\hat{n}}.time$ again gives us:

$$\begin{aligned}(+)\ \ (\ & isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time) \\ & \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time)\ ) \land \\ & ResMapCheck(\sigma_{\hat{n}}.res, v, m)\end{aligned}$$

Regarding the reservation map there are two cases:

$v = \tilde{v} \land src(m) = \tilde{s} \land m.id = \tilde{id} \land \tilde{idx} = m.idx$: The event's guard only deletes the corresponding version of the reservation if holds that

$$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx).expT < \sigma_{\hat{n}}.time$$

By (9) in (+) $ResMapCheck(\sigma_{\hat{n}}.res, v, m)$ it holds:

$$(9)\ res(v, src(m), m.id).vrs(m.idx).expT = m.expT$$

and by assumption $\sigma_{\hat{n}+1}.time \leq m.expT$ it holds

$$m.expT \geq \sigma_{\hat{n}+1}.time = \sigma_{\hat{n}}.time$$

i.e. a contradiction.

Otherwise: The event's actions do not affect the version of the reservation corresponding m, i.e.

$$\begin{aligned}&\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\ &\quad = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)\end{aligned}$$

and therefore the claim follows by (+).

$CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: As in RST it follows (+), hence in particular

$$\begin{aligned}&(\ \mathit{isMrkd}(\sigma_{\hat{n}}.\mathit{res}, v, m, \sigma_{\hat{n}}.\mathit{time})\\ &\quad \vee \mathit{isRsvd}(\sigma_{\hat{n}}.\mathit{res}, v, m, \sigma_{\hat{n}}.\mathit{time})\ ) \wedge\\ &\mathit{ResMapCheck}(\sigma_{\hat{n}}.\mathit{res}, v, m)\end{aligned}$$

The following cases:

$\tilde{m} \approx m \wedge \tilde{m}.ptr = m.ptr \wedge \tilde{v} = v$: By Lemma 4 it follows

$$\begin{aligned}&\mathit{isMrkd}(\sigma_{\hat{n}+1}.\mathit{res}, v, \tilde{m}, \sigma_{\hat{n}+1}.\mathit{time})\\ &\mathit{ResMapCheck}(\sigma_{\hat{n}+1}.\mathit{res}, v, \tilde{m})\end{aligned}$$

Together with $\tilde{m} \approx m$ and $\tilde{m}.ptr = m.ptr$ it follows by Lemma 43 and Lemma 42 that

$$\begin{aligned}&\mathit{isMrkd}(\sigma_{\hat{n}+1}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\\ &\mathit{ResMapCheck}(\sigma_{\hat{n}+1}.\mathit{res}, v, m)\end{aligned}$$

$(\tilde{m} \not\approx m \vee \tilde{m}.ptr \neq m.ptr) \wedge \tilde{v} = v$: $\tilde{m} \not\approx m \vee \tilde{m}.ptr \neq m.ptr$ implies

$$\begin{aligned}&(a)\ \mathit{src}(m) \neq \mathit{src}(\tilde{m}) \vee\\ &(b)\ m.id \neq \tilde{m}.id \vee\\ &(c)\ m.idx \neq \tilde{m}.idx \vee\\ &(d)\ m.path \neq \tilde{m}.path \vee\\ &(e)\ m.\mathit{first} \neq \tilde{m}.\mathit{first} \vee\\ &(e)\ m.\mathit{last} \neq \tilde{m}.\mathit{last} \vee\\ &(g)\ m.minBW \neq \tilde{m}.minBW \vee\\ &(h)\ m.maxBW \neq \tilde{m}.maxBW \vee\\ &(i)\ m.\mathit{expT} \neq \tilde{m}.\mathit{expT} \vee\\ &(j)\ m.ptr \neq \tilde{m}.ptr\end{aligned}$$

Two cases:

$(a) \vee (b) \vee (c)$: The event's action do not affect the version of the reservation corresponding to m, i.e.

$$\begin{aligned}&\sigma_{\hat{n}}.\mathit{res}(v, \mathit{src}(m), m.id).\mathit{vrs}(m.idx)\\ &\quad = \sigma_{\hat{n}+1}.\mathit{res}(v, \mathit{src}(m), m.id).\mathit{vrs}(m.idx)\end{aligned}$$

and the claim follows with (+).

Otherwise: I.e.

$$\begin{aligned}&src(m) = src(\tilde{m}) \land \\ &m.id = \tilde{m}.id \land \\ &m.idx = \tilde{m}.idx\end{aligned}$$

but at least one of the other cases α in $(d), \ldots, (j)$ does not hold.
By $(+)$ holds

$$\begin{aligned}&(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\ &\quad \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \land \\ &ResMapCheck(\sigma_{\hat{n}}.res, v, m)\end{aligned}$$

hence

$$\begin{aligned}&r = \sigma_{\hat{n}}.res(v, src(m), m.id) \land \\ \neg&(0)\ r \neq \perp \land \\ &(1)\ r.path = m.path \land \\ &(2)\ r.ptr = m.ptr \land \\ &(3)\ r.first = m.first \land \\ &(4)\ r.last = m.last \land \\ \neg&(5)\ r.vrs \neq \emptyset \\ \neg&(6)\ r.vrs(m.idx) \neq \perp \land \\ &(7)\ r.minBW = m.minBW \land \\ &(8)\ r.vrs(m.idx).maxBW = m.maxBW \land \\ &(9)\ r.vrs(m.idx).expT = m.expT\end{aligned}$$

But this is in contradiction to the event's guard $ResMapCheck(\sigma_{\hat{n}}.res, v, \tilde{m})$ (using $\tilde{v} = v$) and the case

α that does not hold, i.e.,

$$\begin{aligned}
&\sigma_{\hat{n}}.res(v, src(m), m.id) \\
&\quad = \sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id) \land \\
&\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\
&\quad = \sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id).vrs(\tilde{m}.idx) \land \\
&\sigma_{\hat{n}}.res(v, src(m), m.id).\alpha = m.\alpha \land \\
&\sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id).\alpha = \tilde{m}.\alpha \land \\
&m.\alpha \neq \tilde{m}.\alpha
\end{aligned}$$

Hence this guard is not satisfied and the event could not happen.

$\tilde{v} \neq v$: The event's actions do not affect version of the reservation corresponding m, i.e.

$$\begin{aligned}
&\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\
&\quad = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)
\end{aligned}$$

and therefore the claim follows by (+).

$CRT_D(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: The event does not affect the global time and the reservation map, hence the claim follows by *IH*.

$FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As in RST it follows (+), hence in particular

$$\begin{aligned}
&(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\
&\quad \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \land \\
&ResMapCheck(\sigma_{\hat{n}}.res, v, m)
\end{aligned}$$

The following cases:

$\tilde{m} \approx m \land \tilde{m}.ptr = m.ptr \land \tilde{v} = v$: By Lemma 5 it follows

$$\begin{aligned}
&isMrkd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time) \\
&ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})
\end{aligned}$$

Together with $\tilde{m} \approx m$ and $\tilde{m}.ptr = m.ptr$ it follows by Lemma 43 and Lemma 42 that

$$\begin{aligned}
&isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time) \\
&ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)
\end{aligned}$$

$(\tilde{m} \not\approx m \vee \tilde{m}.ptr \neq m.ptr) \wedge \tilde{v} = v$: $\tilde{m} \not\approx m \vee \tilde{m}.ptr \neq m.ptr$ implies

$$\begin{aligned}
&(a)\ src(m) \neq src(\tilde{m}) \vee \\
&(b)\ m.id \neq \tilde{m}.id \vee \\
&(c)\ m.idx \neq \tilde{m}.idx \vee \\
&(d)\ m.path \neq \tilde{m}.path \vee \\
&(e)\ m.first \neq \tilde{m}.first \vee \\
&(e)\ m.last \neq \tilde{m}.last \vee \\
&(g)\ m.minBW \neq \tilde{m}.minBW \vee \\
&(h)\ m.maxBW \neq \tilde{m}.maxBW \vee \\
&(i)\ m.expT \neq \tilde{m}.expT \vee \\
&(j)\ m.ptr \neq \tilde{m}.ptr
\end{aligned}$$

Two cases:

$(a) \vee (b) \vee (c)$: The event's action do not affect the version of the reservation corresponding to m, i.e.

$$\begin{aligned}
&\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\
&\quad = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)
\end{aligned}$$

and the claim follows with (+).

Otherwise: I.e.

$$\begin{aligned}
&src(m) = src(\tilde{m}) \wedge \\
&m.id = \tilde{m}.id \wedge \\
&m.idx = \tilde{m}.idx
\end{aligned}$$

but at least one of the other cases α in $(d), \ldots, (j)$ does not hold.
By (+) holds

$$\begin{aligned}
&(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\
&\quad \vee isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \wedge \\
&ResMapCheck(\sigma_{\hat{n}}.res, v, m)
\end{aligned}$$

hence with $r = \sigma_{\hat{n}}.res(v, src(m), m.id)$

$$\begin{aligned}
\neg(0)\ & r \neq \bot \wedge \\
(1)\ & r.path = m.path \wedge \\
(2)\ & r.ptr = m.ptr \wedge \\
(3)\ & r.first = m.first \wedge \\
(4)\ & r.last = m.last \wedge \\
\neg(5)\ & r.vrs \neq \emptyset \\
\neg(6)\ & r.vrs(m.idx) \neq \bot \wedge \\
(7)\ & r.vrs(m.idx).minBW = m.minBW \wedge \\
(8)\ & r.vrs(m.idx).maxBW = m.maxBW \wedge \\
(9)\ & r.vrs(m.idx).expT = m.expT
\end{aligned}$$

But this is in contradiction to the event's guard $ResMapCheck(\sigma_{\hat{n}}.res, v, \tilde{m})$ (using $\tilde{v} = v$) and the case α that does not hold, i.e.

$$\begin{aligned}
&\sigma_{\hat{n}}.res(v, src(m), m.id).\alpha = m.\alpha \wedge \\
&\sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id).\alpha = \tilde{m}.\alpha \wedge \\
&m.\alpha \neq \tilde{m}.\alpha
\end{aligned}$$

$\tilde{v} \neq v$: The event's actions do not affect version of the reservation corresponding m, i.e.

$$\begin{aligned}
&\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\
&\quad = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)
\end{aligned}$$

and therefore the claim follows by (+).

$CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As in FWD

$$\tilde{m} \approx m \wedge \tilde{m}.ptr = m.ptr \wedge \tilde{v} = v$$

By using Lemma 6 instead of Lemma 5 it follows

$$\begin{aligned}
&isRsvd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time) \\
&ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})
\end{aligned}$$

instead of

$$\begin{aligned}
&isMrkd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time) \\
&ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})
\end{aligned}$$

$TRN(\tilde{m},\tilde{m}',\tilde{v},\tilde{i},\tilde{t})$: As in CMP.

$UPT(\tilde{m},\tilde{m}',\tilde{v},\tilde{i},\tilde{t})$: As in CMP.

$BWD(\tilde{m},\tilde{m}',\tilde{v},\tilde{i},\tilde{t})$: As in CMP.

$FIN(\tilde{m},\tilde{m}',\tilde{v},\tilde{i},\tilde{t})$:

$CLT(\tilde{m},\tilde{m}',\tilde{a},\tilde{i})$: As in CRT_D.

$ATK(\tilde{m},\tilde{v},\tilde{i})$: As in CRT_D.

$RMV(\tilde{m},\tilde{m}',\tilde{v},\tilde{i},\tilde{t})$: As in RST it follows (+), hence in particular

$$\begin{aligned}&(\ \mathit{isMrkd}(\sigma_{\hat{n}}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\\ &\quad \vee \mathit{isRsvd}(\sigma_{\hat{n}}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\ ) \wedge\\ &\mathit{ResMapCheck}(\sigma_{\hat{n}}.\mathit{res}, v, m)\end{aligned}$$

Two cases:

$\tilde{m} \sim m \wedge \tilde{v} = v$: The event's action *remove* only sets the field $resBW$ of the version corresponding to $\tilde{m}$ (and in this case m) to $\perp$, but keeps the other fields of $\sigma_{\hat{n}}.\mathit{res}$ the same. Hence it holds

$$\begin{aligned}&\mathit{isMrkd}(\sigma_{\hat{n}}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\\ &\quad \Rightarrow \mathit{isMrkd}(\sigma_{\hat{n}+1}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\\ &\mathit{isRsvd}(\sigma_{\hat{n}}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\\ &\quad \Rightarrow \mathit{isMrkd}(\sigma_{\hat{n}+1}.\mathit{res}, v, m, \sigma_{\hat{n}+1}.\mathit{time})\\ &\mathit{ResMapCheck}(\sigma_{\hat{n}}.\mathit{res}, v, m)\\ &\quad \Rightarrow \mathit{ResMapCheck}(\sigma_{\hat{n}+1}.\mathit{res}, v, m)\end{aligned}$$

and therefore the claim follows by (+).

Otherwise: The event's actions do not affect version of the reservation corresponding m, i.e.

$$\begin{aligned}&\sigma_{\hat{n}}.\mathit{res}(v, \mathit{src}(m), m.id).\mathit{vrs}(m.idx)\\ &\quad = \sigma_{\hat{n}+1}.\mathit{res}(v, \mathit{src}(m), m.id).\mathit{vrs}(m.idx)\end{aligned}$$

and therefore the claim follows by (+).

$DST(\tilde{m},\tilde{m}',\tilde{v},\tilde{i},\tilde{t})$: As in RMV.

$DRP(\tilde{m},\tilde{v},\tilde{i},\tilde{t})$: As in CRT_D.

□

B.7 Prefix Lemmata

Lemma 45.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = CRT_R(m, v, i, t) \ \wedge\ nodes(m) \subseteq H \Rightarrow\\
&\quad\ \forall k \in [0; m.ptr].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad\ \ \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \wedge\ m \approx \tilde{m} \ \wedge\ k = \tilde{m}.ptr \ \wedge\\
&\quad\ \ m.accBW[k - m.first - 1] = \tilde{m}.accBW[k - m.first - 1] \wedge\\
&\quad\ \ \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \ \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \ \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \ \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Note that by definition $nil[k] = \bot$ and that by the guards of CRT_R it holds $m.ptr = 0$, i.e., the premises of the last three constraints are not satisfied and therefore the constraints hold trivially.

Proof. This holds trivially for $\tilde{n} = n$ and therefore by assumption

$$CRT_R(m, v, i, t) = \lambda_n = \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t}).$$

This implies

$$\begin{aligned}
&\sigma_{\tilde{n}}.time = \sigma_n.time = t \in [m.expT - maxT, t]\\
&k = \tilde{m}.ptr = 0
\end{aligned}$$

and by $m = \tilde{m}$ also $m \approx \tilde{m}$ and $m.accBW = \tilde{m}.accBW = nil$. □

Lemma 46.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = FWD(m, m', v, i, t) \ \wedge\ nodes(m) \subseteq H \Rightarrow\\
&\quad\ \forall k \in [0; m.ptr].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad\ \ \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \wedge\ m \approx \tilde{m} \ \wedge\ k = \tilde{m}.ptr \ \wedge\\
&\quad\ \ m.accBW[k - m.first - 1] = \tilde{m}.accBW[k - m.first - 1] \wedge\\
&\quad\ \ \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \ \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \ \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \ \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Proof. Similarly to the proof of Lemma 47. □

Lemma 47.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = CMP(m, m', v, i, t) \ \wedge \ nodes(m) \subseteq H \Rightarrow\\
&\quad \forall k \in [0; m.ptr].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad\ \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \wedge \ m \approx \tilde{m} \ \wedge \ k = \tilde{m}.ptr \ \wedge\\
&\quad\ m.accBW[k - m.first - 1] = \tilde{m}.accBW[k - m.first - 1] \wedge\\
&\quad\ \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad\ \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Proof. Assume $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in V$, $i \in I$, $t \in \mathbb{N}$ with (1) $\lambda_n = CMP(m, m', v, i, t)$ and (2) $nodes(m) \subseteq H$.

We prove the statement by induction on $d = m.ptr - m.first$. Note that by the event's guards of CMP it holds that $m.ptr \geq m.first$.

$d = 0$: i.e., $m.ptr = m.first$. There are two cases:

$k = m.ptr$: the claim holds trivially by (1).

$k < m.ptr$: Given (1), it follows by Lemma 32 it follows

$$\begin{aligned}
&m.ptr = m.first \Rightarrow\\
&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = FWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t}) \wedge\\
&\quad m = \bar{m}' \ \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

Hence, with $\tilde{n} = \bar{n}$ and $\tilde{m} = \bar{m} \approx \bar{m}' = m$ (i.e., $m.expT = \tilde{m}.expT$ and $\tilde{m}.ptr = m.ptr - 1 = m.first - 1$) By applying Lemma 46 to $\lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$ it follows

$$\begin{aligned}
&\forall k \in [0; \tilde{m}.ptr].\ \exists \bar{n} \leq \tilde{n}, \bar{m}, \bar{m}' \in \mathcal{M}_R \bar{v} \in V, \bar{i} \in I. \\
&\sigma_{\bar{n}}.time \in [\tilde{m}.expT - maxT, \tilde{t}] \ \land\ \tilde{m} \approx \bar{m} \ \land\ k = \bar{m}.ptr \ \land \\
&\tilde{m}.accBW[k - \tilde{m}.first - 1] = \bar{m}.accBW[k - \tilde{m}.first - 1] \ \land \\
&\big(k = 0 \Rightarrow \lambda_{\bar{n}} = CRT_R(\bar{m}, \bar{v}, \bar{i}, \bar{t})\big) \ \land \\
&\big(0 \leq k < \tilde{m}.first \Rightarrow \lambda_{\bar{n}} = FWD(\bar{m}, \bar{m}', \bar{v}, \bar{i}, \bar{t})\big) \ \land \\
&\big(\tilde{m}.first \leq k < \tilde{m}.last \Rightarrow \lambda_{\bar{n}} = CMP(\bar{m}, \bar{m}', \bar{v}, \bar{i}, \bar{t})\big) \ \land \\
&\big(k = \tilde{m}.last \Rightarrow \lambda_{\bar{n}} = TRN(\bar{m}, \bar{m}', \bar{v}, \bar{i}, \bar{t})\big).
\end{aligned}$$

where $m \approx \tilde{m} \approx \bar{m}$ and therefore $\tilde{m}.expT = m.expT$, $\tilde{m}.first = m.first$, and $\tilde{m}.last = m.last$. Furthermore, it holds $\bar{n} \leq \tilde{n} \leq n$ and therefore $\tilde{t} \leq t$ and therefore the claim for $k \leq k \in \tilde{m}.ptr = m.first - 1$.

$d \to d+1$: Assume the *IH* for $d = m.ptr - m.first$ for any $m \in \mathcal{M}$, i.e., it holds:

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\ d = m.ptr - m.first \ \land \\
&\quad \lambda_n = CMP(m, m', v, i, t) \ \land\ nodes(m) \subseteq H \Rightarrow \\
&\quad\ \forall k \in [0; m.ptr].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I. \\
&\quad\ \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \land\ m \approx \tilde{m} \ \land\ k = \tilde{m}.ptr \ \land \\
&\quad\ m.accBW[k - m.first - 1] = \tilde{m}.accBW[k - m.first - 1] \ \land \\
&\quad\ \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \ \land \\
&\quad\ \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \land \\
&\quad\ \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \land \\
&\quad\ \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Let $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in V$, $i \in I$, $t \in \mathbb{N}$ with $d + 1 = m.ptr - m.first$.

Assume $(*)$ $\lambda_n = CMP(m, m', v, i, t)$ and $nodes(m) \subseteq H$.

Given $(*)$ it follows by the event's guards and $d > 0$ that $m.\mathit{first} < m.ptr < m.\mathit{last}$ and therefore by Lemma 31

$$\begin{aligned}
&m.\mathit{first} < m.ptr < m.\mathit{last} \Rightarrow\\
&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = CMP(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t}) \wedge\\
&\quad m = \bar{m}'
\end{aligned}$$

Therefore, by $(+)$ $\lambda_{\bar{n}} = CMP(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})$, $\bar{m} \approx m$, $d+1 = m.ptr - m.\mathit{first}$ and $\bar{m}.ptr = m.ptr - 1$, it follows

$$\bar{m}.ptr - \bar{m}.\mathit{first} = m.ptr - 1 - m.\mathit{first} = d + 1 - 1 = d$$

Therefore we can apply *IH* to $(+)$ and obtain

$$\begin{aligned}
&\forall k \in [0; \bar{m}.ptr].\ \exists \tilde{n} \leq \bar{n}, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\sigma_{\tilde{n}}.time \in [\bar{m}.expT - maxT, \bar{t}] \ \wedge\ \bar{m} \approx \tilde{m}\ \wedge\ k = \tilde{m}.ptr \wedge\\
&\bar{m}.accBW[k - \bar{m}.\mathit{first} - 1] = \tilde{m}.accBW[k - \bar{m}.\mathit{first} - 1] \wedge\\
&\big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \wedge\\
&\big(0 \leq k < \bar{m}.\mathit{first} \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \wedge\\
&\big(\bar{m}.\mathit{first} \leq k < \bar{m}.\mathit{last} \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \wedge\\
&\big(k = \bar{m}.\mathit{last} \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Since $\bar{m} \approx m$ (i.e., $m.expT = \bar{m}.expT$, $m.\mathit{first} = \bar{m}.\mathit{first}$ and $m.\mathit{last} = \bar{m}.\mathit{last}$) and $\bar{m}.ptr = m.ptr-1$ the claim follows for all $k \in [0; m.ptr-1]$.

The case for $k = m.ptr$ follows directly from assumption (1).

$\square$

Lemma 48.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = TRN(m, m', v, i, t) \ \wedge\ nodes(m) \subseteq H \Rightarrow\\
&\quad \forall k \in [0; m.ptr].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \wedge\ m \approx \tilde{m} \ \wedge\ k = \tilde{m}.ptr \ \wedge\\
&\quad m.accBW[k - m.first - 1] = \tilde{m}.accBW[k - m.first - 1] \wedge\\
&\quad \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Proof. Assume $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in V$, $i \in I$, $t \in \mathbb{N}$ with (1) $\lambda_n = TRN(m, m', v, i, t)$ and (2) $nodes(m) \subseteq H$.

Given (1) it follows by Lemma 30

$$\begin{aligned}
&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = CMP(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge\\
&\quad m = \bar{m}'
\end{aligned}$$

Hence, for $k = m.ptr = m.last$ and $m = \bar{m}' \approx \bar{m}$ the claim follows.

Furthermore, given $\lambda_{\bar{n}} = CMP(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})$ it follows by Lemma 47

$$\begin{aligned}
&\forall k \in [0; \bar{m}.ptr].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad \sigma_{\tilde{n}}.time \in [\bar{m}.expT - maxT, \bar{t}] \ \wedge\ \bar{m} \approx \tilde{m} \ \wedge\ k = \tilde{m}.ptr \ \wedge\\
&\quad \bar{m}.accBW[k - \bar{m}.first - 1] = \tilde{m}.accBW[k - \bar{m}.first - 1] \wedge\\
&\quad \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(0 \leq k < \bar{m}.first \Rightarrow \lambda_{\tilde{n}} = FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(\bar{m}.first \leq k < \bar{m}.last \Rightarrow \lambda_{\tilde{n}} = CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(k = \bar{m}.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Since $\bar{m}.ptr = m.ptr - 1$ and $m = \bar{m}' \approx \bar{m} \approx \tilde{m}$ it follows the claim for all $k \in [0; m.ptr - 1]$. □

Lemma 49.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = TRN(m, m', v, i, t) \,\wedge\, nodes(m) \subseteq H \Rightarrow\\
&\quad\; \forall k \in [m.ptr; m.last].\, \exists \tilde{n} \le n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad\;\; \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \,\wedge\, m \approx \tilde{m} \,\wedge\, k = \tilde{m}.ptr \,\wedge\\
&\quad\;\; m.accBW = \tilde{m}.accBW \,\wedge\\
&\quad\;\; \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \,\wedge\\
&\quad\;\; \big(0 \le k < m.first \Rightarrow \lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \,\wedge\\
&\quad\;\; \big(m.first \le k < m.last \Rightarrow \lambda_{\tilde{n}} = UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \,\wedge\\
&\quad\;\; \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Note that by the guards of event *TRN* it follows that $m.ptr = m.last$ and therefore the premises of the three constraints with $0 \le k < m.last$ are not satisfied and the constraints hold trivially.

Proof. By the event's guard of TRN it holds that $k = m.ptr = m.last$. Setting $\tilde{n} = n$ and $\tilde{m} = m$ the claim follows trivially by assumption $\lambda_n = TRN(m, m', v, i, t)$. □

Lemma 50.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = UPT(m, m', v, i, t) \,\wedge\, nodes(m) \subseteq H \Rightarrow\\
&\quad\; \forall k \in [m.ptr; m.last].\, \exists \tilde{n} \le n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad\;\; \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \,\wedge\, m \approx \tilde{m} \,\wedge\, k = \tilde{m}.ptr \,\wedge\\
&\quad\;\; m.accBW = \tilde{m}.accBW \,\wedge\\
&\quad\;\; \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \,\wedge\\
&\quad\;\; \big(0 \le k < m.first \Rightarrow \lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \,\wedge\\
&\quad\;\; \big(m.first \le k < m.last \Rightarrow \lambda_{\tilde{n}} = UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \,\wedge\\
&\quad\;\; \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Proof. By induction over $d = m.last - m.ptr$, similarly to the proof of Lemma 47. □

Lemma 51.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = BWD(m, m', v, i, t) \ \wedge \ nodes(m) \subseteq H \Rightarrow\\
&\quad \forall k \in [m.ptr; m.last].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \wedge \ m \approx \tilde{m} \ \wedge \ k = \tilde{m}.ptr \ \wedge\\
&\quad m.accBW = \tilde{m}.accBW \ \wedge\\
&\quad \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Proof. By induction over $d = m.first - m.ptr$, similarly to the proof of Lemma 47. □

Lemma 52.

$$\begin{aligned}
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.\\
&\quad \lambda_n = FIN(m, m', v, i, t) \ \wedge \ nodes(m) \subseteq H \Rightarrow\\
&\quad \forall k \in [m.ptr; m.last].\ \exists \tilde{n} \leq n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\quad \sigma_{\tilde{n}}.time \in [m.expT - maxT, t] \ \wedge \ m \approx \tilde{m} \ \wedge \ k = \tilde{m}.ptr \ \wedge\\
&\quad m.accBW = \tilde{m}.accBW \ \wedge\\
&\quad \big(k = 0 \Rightarrow \lambda_{\tilde{n}} = FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(0 \leq k < m.first \Rightarrow \lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(m.first \leq k < m.last \Rightarrow \lambda_{\tilde{n}} = UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big) \ \wedge\\
&\quad \big(k = m.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

Proof. Assume (1) $\lambda_n = FIN(m, m', v, e, t)$ and (2) $nodes(m) \subseteq H$.
With $\tilde{n} = n$ and $\tilde{m} = m$ the claim follows for (+) $k = \tilde{m}.ptr = m.ptr$.
Furthermore, applying Lemma 25 to (1) and (2) implies

$$\begin{aligned}
&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\
&\quad \lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t}) \wedge\\
&\quad m = \bar{m}' \wedge\\
&\quad \forall \hat{n}.\ \bar{n} < \hat{n} \leq n \Rightarrow m \in \sigma_{\hat{n}}.buf(v, e)
\end{aligned}$$

with $\bar{m}.ptr = m.ptr + 1$.

Applying Lemma 51 to $\lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})$ implies

$$\begin{aligned}
&\forall k \in [\bar{m}.ptr; \bar{m}.last].\ \exists \tilde{n} \leq \bar{n}, \tilde{m}, \tilde{m}' \in \mathcal{M}_R \tilde{v} \in V, \tilde{i} \in I.\\
&\sigma_{\tilde{n}}.time \in [\bar{m}.expT - maxT, t] \ \wedge\ \bar{m} \approx \tilde{m}\ \wedge\ k = \tilde{m}.ptr\ \wedge\\
&\bar{m}.accBW = \tilde{m}.accBW\ \wedge\\
&\big(k = 0 \Rightarrow \lambda_{\tilde{n}} = FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big)\ \wedge\\
&\big(0 \leq k < \bar{m}.first \Rightarrow \lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big)\ \wedge\\
&\big(\bar{m}.first \leq k < \bar{m}.last \Rightarrow \lambda_{\tilde{n}} = UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big)\ \wedge\\
&\big(k = \bar{m}.last \Rightarrow \lambda_{\tilde{n}} = TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})\big).
\end{aligned}$$

By $m \approx \tilde{m} \approx \bar{m}$ (i.e., $m.expT = \bar{m}.expT$, etc) and $m.accBW = \tilde{m}.accBW = \bar{m}.accBW$ the claim follows for $k \in [\bar{m}.ptr; \bar{m}.last] = [m.ptr + 1; m.last]$ and therefore together with (+) for $k \in [m.ptr; m.last]$. □

B.8 Theorem 1

Theorem 27. *If an AS s makes a successful reservation m at time t, then all ASes on m's path segment added their* avail *and* ideal *computations to $m.accBW$ and reserve $finBW(m)$ until it expires*

$$\begin{aligned}
&\forall m \in \mathcal{M}_R, t \in \mathbb{N}.\ SuccRes(m, t) \Rightarrow\\
&\qquad \forall n \in \mathbb{N}, v \in V, vrs \in VrsMap, k \in [m.first; m.last].\\
&\qquad\quad \sigma_n.time \in\ ]t; m.expT]\ \wedge\ v = m.path[k].as\ \wedge\\
&\qquad\quad vrs = \sigma_n.res(v, src(m), m.id).vrs(m.idx)\\
&\qquad\quad \Rightarrow cvalid(vrs, \sigma_n.time) \wedge vrs = finBW(m)\ \wedge\\
&\qquad\qquad \exists \tilde{n} < n, \tilde{m} \in \mathcal{M}_R.\ \tilde{m} \approx m\ \wedge\\
&\qquad\qquad\quad m.accBW[k - m.first] = (\!|\ avBW = \text{avail}(\tilde{m}, \sigma_{\tilde{n}}.res);\\
&\qquad\qquad\qquad\qquad\qquad\qquad idBW = \text{ideal}(\tilde{m}, \sigma_{\tilde{n}}.res)\ |\!).
\end{aligned}$$

Proof. First we prove that

$$\begin{aligned}
&\forall n \in \mathbb{N}, v \in V, k \in [m.first; m.last].\\
&\quad \sigma_n.time \in\ ]t; m.expT]\ \wedge\ v = m.path[k].as\\
&\quad \Rightarrow \sigma_n.res(v, s, m.id).vrs(m.idx).resBW = finBW(m).
\end{aligned}$$

Given Assumption CF, we show that the FIN event is preceded by an BWD event on the previous AS. Given a messages $m \in buf(v, i)$ with

$v \in nodes(m)$, we can show by induction that there was corresponding message processing event in the time interval $[m.expT - maxT, t]$ with a corresponding message $\tilde{m}$

$$\begin{aligned}
&\forall v \in nodes(m).\ \exists evt \in \{CRT_R, FWD, CMP, TRN, UPT, BWD\}.\\
&\ \exists \tilde{n} \in \mathbb{N}, \tilde{m}, \tilde{m}' \in \mathcal{M}_R, \tilde{i} \in I, \tilde{t} \in \mathbb{N}.\\
&\ \sigma_{\tilde{n}}.time < t \wedge \lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', v, \tilde{i}, \tilde{t}) \wedge \tilde{m} \approx m
\end{aligned}$$

In the case distinction we need to exclude that the attacker put m into the buffer by using Lemma 11 which is based on our Assumption HP.

Next, we show that each of these events is unique for each successful reservation with message m

$$\begin{aligned}
&\forall \tilde{n}, \hat{n} \in \mathbb{N}, evt \in \{CRT_R, FWD, CMP, TRN, UPT, BWD\},\\
&\forall \tilde{m}, \tilde{m}', \hat{m}, \hat{m}' \in \mathcal{M}_R, v \in H, \tilde{i}, \hat{i} \in I, \tilde{t}, \hat{t} \in \mathbb{N}.\\
&\lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', v, \tilde{i}, \tilde{t}) \wedge \lambda_{\hat{n}} = evt(\hat{m}, \hat{m}', v, \hat{i}, \hat{t}) \wedge\ \tilde{m} \approx \hat{m}\\
&\ \Rightarrow \tilde{n} = \hat{n}
\end{aligned}$$

and therefore exclude that there was another corresponding message processing event that changes the reservation done by $\tilde{m}$.

Using Assumption nDE, we can also exclude that the reservation gets deleted in the time interval $[m.expT - maxT, t]$.

Thus, for each successful reservation with a message m, we obtain a list of transitions $l_m = [\lambda_1, \ldots, \lambda_{2 \cdot m.last}]$ ordered by time, where each transition corresponds to the message unique processing event for m. Given l_m we can finally show by induction on its length

$$\begin{aligned}
&\exists \tilde{n} < n, \tilde{m} \in \mathcal{M}_R.\ \tilde{m} \approx m\ \wedge\\
&\ m.accBW[k - m.first] = (\!|\ avBW = avail(\tilde{m}, \sigma_{\tilde{n}}.res);\\
&\ m.accBW[k - m.first] = (\!|\ idBW = ideal(\tilde{m}, \sigma_{\tilde{n}}.res)\ |\!).
\end{aligned}$$

□

B.9 Theorem 2

Theorem 28. *If there are constant demands $\mathcal{D}$ between t_0 and t_1, then after time $t_0 + 2maxT$ all reservations allocate the ideal bandwidth until t_1, i.e.,*

$$\begin{aligned}&\exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + 2maxT \land \\ &\forall m \in rng(\mathcal{D}), r \in Res, n > \tilde{n}, v \in sgmt(m). \\ &\ \sigma_n.time \in]t_0 + 2maxT; t_1] \ \land \ r = \sigma_n.res_v(src(m), m.id) \\ &\ \ \Rightarrow allocBW(r, \sigma_n.time) = \min_{x \in sgmt(m)}\{\text{ideal}(m, \sigma_{\tilde{n}}.res_x)\}\end{aligned}$$

Proof. We first show that after *maxT* the requested demands in all reservation maps correspond to $\mathcal{D}$,

$$\forall n \in \mathbb{N}. \sigma_n \in]t_0 + maxT; t_1].\ \sigma_n \vdash \mathcal{D}$$

From this follows that for any honest AS v and reservation r corresponding to a reservation message $m \in rng(\mathcal{D})$ the function *demBW* remains constant and evaluates to $m.maxBW$

$$\begin{aligned}&\forall n \in \mathbb{N}, v \in H, m \in rng(\mathcal{D}), r \in Res. \\ &\ \sigma_n.time\ ]t_0 + maxT; t_1] \land r = \sigma_n.res_v(src(m), m.id) \\ &\ \ \Rightarrow demBW(r, t) = m.maxBW.\end{aligned}$$

Since *ideal* bandwidth computation on first AS of a path only depends on the value of the function $demBW_v$ and not on *ideal* bandwidth computations executed at previous ASes on the path, it remains constant as well.

By induction on the length of the message's path we can show that the *ideal* computations for all ASes on the path remain constant.

Next, assuming that all *ideal* computations remain constant, we show that the *avail* computation is greater than the *ideal* computation for every renewal

$$\begin{aligned}&\forall n \in \mathbb{N}, evt \in \{CMP, UPT, TRN\}. \\ &\forall v \in H, m, m' \in \mathcal{M}_R, i \in I, t \in \mathbb{N} \\ &\ \sigma_n.time \in]t_0 + maxT; t_1] \land \lambda_n = evt(m, m', v, i, t) \land m \vdash \mathcal{D} \\ &\ \ \Rightarrow ideal(m, \sigma_n.res_v) \geq avail(m, \sigma_n.res_v)\end{aligned}$$

and together with the definition of *finBW*

$$finBW(m) = \min\{m.maxBW, \min(m.accBW)\}$$

it follows that

$$allocBW\,(r, \sigma_n.time) = \min_{x \in sgmt(m)}\{ideal(m, \sigma_{\tilde{n}}.res_x)\}$$

$\square$

B.10 Properties of N-Tube's Bandwidth Allocation Computation

B.10.1 Positivity

The functions *avail* and *ideal* always compute strictly positive values, hence, for a valid request with $(*)$ $m.minBW = 0$ and $(+)$ $nodes(m) \subseteq H$ a positive amount of bandwidth is always allocated:

$$\begin{aligned}
&\forall evt \in \{CMP, TRN, UPT\}.\\
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N}.\\
&\ \lambda_n = evt(m, m', v, i, t) \Rightarrow finBW(m, resM_v) > 0.
\end{aligned}$$

Proof. W.l.o.g. we show the claim for the event UPT. For the events TRN and CMP the proof works analogously. By induction on n:

$n = 0$: Since in a valid execution all buffers are empty in $\sigma_0.buf$ and therefore no message processing event can happen, the premise $\lambda_n = UPT(m, m', v, i, t)$ is not satisfied and the claim holds trivially.

$n > 0$: By *IH* it holds

$$\begin{aligned}
&\forall n' < n, m, m' \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N}.\\
&\ \lambda_{n'} = UPT(m, m', v, i, t) \Rightarrow finBW(m, resM_v) > 0.
\end{aligned}$$

Given AS v, message m with $(i, v, e) = cur(m)$. By the event's guard it holds that m is valid, in particular, that

$$\begin{aligned}
&(1)\ \ m.maxBW > 0,\\
&(2)\ \ cap(v, e) > 0,\\
&(3)\ \ cap(v, i) > 0.
\end{aligned}$$

Furthermore, by the event's action it holds that

```
$save(v \in V, resM \in ResMap, m' \in \mathcal{M}_R) =$
  **let**
   $finBW = \min(m'.maxBW, \min(m'.accBW))$
   $vrs' = (\!| minBW := m'.minBW;$
              $maxBW := m'.maxBW;$
              $idBW := m'.accBW.[m'.ptr - 1].idBW;$
              $resBW := finBW;$
              $expT := m'.expT |\!)$
   $vrsM' = resM(v, src(m'), m'.id).vrs\big(m'.idx \mapsto vrs'\big)$
   $res' = (\!| path := m'.path;$
              $ptr := m'.ptr;$
              $first := m'.first;$
              $last := m'.last;$
              $vrs := vrsM' |\!)$
  **in**
   $resM\big((v, src(m'), m'.id) \mapsto res'\big).$
```

and the event's guards *onPth*, i.e., $m.first < m.ptr < m.last$ and *ResMapCheck* it follows for the reservation corresponding to m, with $r = \sigma_{n+1}.res(v, src(m), m.id)$, that

$$(4)\quad r.first < r.ptr < r.last$$
$$(5)\quad r.vrs(m.idx).expT > t$$

By the function *compute*

```
$compute(m \in \mathcal{M}_R, res \in ResMap, \delta \in ]0;1[, t \in \mathbb{N}) =$
 **let**
  $newBW = (\!| avBW := avail(m, res, \delta, t);$
              $idBW := ideal(m, res, \delta, t) |\!)$
 **in**
  $m(\!| accBW := newBW \# m.accBW |\!).$
```

it follows that

$$\begin{aligned}&m'.accBW[m'.ptr] = \\ &\quad (\!|\ avBW := avail(m, res, \delta, t); idBW := ideal(m, res, \delta, t)\ |\!)\end{aligned}$$

Since $m'.maxBW = m.maxBW$ and (1) it suffices to show that both *avail* and *ideal* return a positive values.

avail : By definition

$$\begin{aligned}&avail(m, resM, \delta, t) = \\ &\textbf{let} \\ &\quad (\!|\ i, v, e\ |\!) = cur(m) \\ &\quad resM' = resM\,((v, src(m), m.id) \mapsto \bot) \\ &\quad resM'_v = filter(resM', v) \\ &\textbf{in} \\ &\delta \cdot \left(cap(v, e) - \sum_{\substack{r \in rng\left(resM'_v\right): \\ resEg(r) = e}} allocBW(r.vrs, t) \right)\end{aligned}$$

By Lemma B.10.1 it follows that

$$cap(v, e) > \sum_{\substack{r \in rng\left(resM'_v\right): \\ resEg(r) = e}} allocBW(r.vrs, t)$$

Note, that in Lemma B.10.1 the removal of all versions of reservation $(v, src(m), m.id)$

$$resM' = resM\,((v, src(m), m.id) \mapsto \bot)$$

is not assumed. By $\delta > 0$ it follows that $avail(m, resM, \delta, t) > 0$.

ideal : By definition it suffices to show that each of the three factors *reqRatio*, *linkRatio*, and *tubeRatio* is positive. Since their denominators are sums of non-negative summands it is sufficient to show that each nominator is positive. We show this only for the factor *reqRatio*, since the other cases can be shown with analogous arguments.

The nominator of *reqRatio* (analogously for $reqRatio_{start}$)

$$\begin{aligned}&reqRatio_{transit}(v, s, id, i, resM, t) = \\ &\quad \frac{adjIdDem(v, resM(v, s, id), resM, t)}{transitDem(v, i, resM, t)}\end{aligned}$$

is given by

$$\mathit{adjIdDem}(v, r, \mathit{resM}, t) = \\ \mathit{egScalFctr}(v, s, e, \mathit{resM}, t) \cdot \min\{\mathit{cap}(v, i), \mathit{cap}(v, e), \mathit{idBW}(r.\mathit{vrs}, t)\}$$

which by (2) and (3) is positive if *egScalFctr* and *idBW* are positive.

egScalFctr : By the definition of *egScalFctr*

$$\mathit{egScalFctr}(v, s, e, \mathit{resM}, t) = \\ \frac{\min\left(\mathit{cap}(v, e), \mathit{egDem}(v, s, e, \mathit{resM}, t)\right)}{\mathit{egDem}(v, s, e, \mathit{resM}, t)}$$

and assumption (2) it suffices to show that $\mathit{egDem}(v, s, e, \mathit{resM}, t)$ is positive. By the definition of *egDem*

$$\mathit{egDem}(v, s, e, \mathit{resM}, t) = \\ \sum_{\substack{r' \in \mathit{rng}(\mathit{resM}): \\ \mathit{resSr}(r')=s \\ \mathit{resEg}(r')=e}} \mathit{reqDem}(v, r', \mathit{resIn}(r'), e, t).$$

it suffices to show that $\mathit{reqDem}(v, r, i, e, t)$ is positive. By the definition of *reqDem*

$$\mathit{reqDem}(v, r, i, e, t) = \\ \min\{\mathit{cap}(v, i), \mathit{cap}(v, e), \mathit{demBW}(r.\mathit{vrs}, t)\}.$$

and (2) and (3) it suffices to show that $\mathit{demBW}(r.\mathit{vrs}, t)$ is positive. By definition of *demBW*

$$\mathit{demBW}(\mathit{vrsM}, t) = \\ \max_{\substack{\mathit{vrs} \in \\ \mathit{rng}(\mathit{vrsM})}} \{\mathit{vrs}.\mathit{maxBW} \mid \mathit{vrs}.\mathit{minBW} \leq \mathit{vrs}.\mathit{resBW} \wedge \mathit{vrs}.\mathit{expT} \geq t\}$$

it suffices to show that

$$\begin{aligned} &(a) \quad r.\mathit{vrs}(m'.\mathit{idx}).\mathit{maxBW} > 0 \\ &(b) \quad r.\mathit{vrs}(m'.\mathit{idx}).\mathit{minBW} \leq r.\mathit{vrs}(m'.\mathit{idx}).\mathit{resBW} \\ &(c) \quad r.\mathit{vrs}(m'.\mathit{idx}).\mathit{expT} \geq t \end{aligned}$$

The fact (a) follows from (1).

The fact (b) follows by assumption $(*)$ and that

$$r.vrs(m'.idx).minBW = m'.minBW = m.minBW = 0$$

The fact (c) follows by (5), $n' < n$, and Lemma 3.

idBW : By the definition of function *idBW*

$$idBW(vrsM, t) = \max_{\substack{vrs\in\\ rng(vrsM)}} \{vrs.idBW \mid vrs.minBW \leq vrs.resBW \wedge vrs.expT \geq t\}.$$

it suffices to show that

$$\begin{aligned} (a)&\quad r.vrs(m'.idx).idBW > 0\\ (b)&\quad r.vrs(m'.idx).minBW \leq r.vrs(m'.idx).resBW\\ (c)&\quad r.vrs(m'.idx).expT \geq t \end{aligned}$$

The fact (a) follows by the definition of the function *save*, in particular by

$$r.vrs(m'.idx).idBW := m'.accBW.[m'.ptr - 1].idBW,$$

By Lemma 28 and assumption $(+)$, there is a $n' < n$ such that

$$\lambda_{n'} = UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$$

and therefore

$$m'.accBW.[m'.ptr - 1].idBW = ideal(\tilde{m}', \sigma_{n'}.res, \delta, \tilde{t})$$

By *IH* it follows that

$$finBW(\tilde{m}, resM_{\tilde{v}}) > 0$$

and therefore in particular

$$ideal(\tilde{m}', \sigma_{n'}.res, \delta, \tilde{t}) > 0$$

The facts (b) and (c) follow analogously to the case above.

Observe that all three factors contain their nominator as a summand in the denominator. Since all the denominators' summands are non-negative, it follows trivially that all three factors are less or equal than 1 and therefore

$$ideal(m, resM, \delta, t) \leq \delta \cdot cap(v, e).$$

□

Lemma.

$$\begin{aligned}&\forall n \in \mathbb{N}, s, v \in H, e \in I, id \in \mathbb{N}, t \in \mathbb{N}.\\ &resM'_v = \mathit{filter}\,(\sigma_n.res(v,s,id), v) \;\wedge\; cap(v,e) > 0\\ &\Rightarrow cap(v,e) > \sum\nolimits_{\substack{r \in rng\left(resM'_v\right):\\ resEg(r)=e}} allocBW(r.vrs, t)\end{aligned}$$

Proof. By induction on n.

$n = 0$: The inequality holds trivially since in the initial state $\sigma_0.res = \emptyset$, hence $allocBW(r.vrs, t) = 0$ for any $r \in rng\left(resM'_v\right)$. By assumption $cap(v,e) > 0$ the claim follows.

$n \rightarrow n+1$: Assume $s, v \in H, e \in I, id \in \mathbb{N}, t \in \mathbb{N}$ with

$$resM'_v = \mathit{filter}\,(\sigma_n(v,s,id).res, v) \wedge cap(v,e) > 0$$

By *IH*

$$\begin{aligned}&\forall s, v \in H, e \in I, id \in \mathbb{N}, t \in \mathbb{N}.\\ &resM'_v = \mathit{filter}\,(\sigma_n(v,s,id).res, v) \wedge cap(v,e) > 0\\ &\Rightarrow cap(v,e) > \sum\nolimits_{\substack{r \in rng\left(resM'_v\right):\\ resEg(r)=e}} allocBW(r.vrs, t)\end{aligned}$$

By case distinction on λ_n. The relevant events are the following:

$FWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: By the event's guard $onWay(m)$, it follows that the reservation that gets updated is filtered out by *filter*

$$\begin{aligned}&\mathit{filter}(resM, v) =\\ &\;\lambda(s', id').\\ &\;\;\mathbf{let}\ r = resM(v, s', id')\\ &\;\;\mathbf{in}\ \left(\mathbf{if}\ r.\mathit{first} \leq r.ptr \leq r.last\ \mathbf{then}\ resM(v, s', id')\ \mathbf{else}\ \perp\right)\end{aligned}$$

Hence, $allocBW(r.vrs, t)$ stays the same the claim follows by *IH*.

$CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: The only relevant case is if $\tilde{v} = v$. Then by the event's action

```
$save(v \in V, resM \in ResMap, m' \in \mathcal{M}_R) =$
  **let**
  $finBW = \min(m'.maxBW, \min(m'.accBW))$
  $vrs' = \llparenthesis\, minBW := m'.minBW;$
           $maxBW := m'.maxBW;$
           $idBW := m'.accBW.[m'.ptr - 1].idBW;$
           $resBW := finBW;$
           $expT := m'.expT \,\rrparenthesis$
  $vrsM' = resM(v, src(m'), m'.id).vrs\left(m'.idx \mapsto vrs'\right)$
  $res' = \llparenthesis\, path := m'.path;$
           $ptr := m'.ptr;$
           $first := m'.first;$
           $last := m'.last;$
           $vrs := vrsM' \,\rrparenthesis$
  **in**
  $resM\left((v, src(m'), m'.id) \mapsto res'\right).$
```

and the event's guards *onPth*, i.e., $\tilde{m}.first < \tilde{m}.ptr < \tilde{m}.last$ and *ResMapCheck* it follows for the reservation corresponding to $\tilde{m}$, with $r = \sigma_{n+1}.res(v, src(\tilde{m}), \tilde{m}.id)$, that

$$(4)\quad r.first < r.ptr < r.last$$

$$(5)\quad r.vrs(\tilde{m}.idx).expT > t$$

By *IH* it holds that

$$resM_v = filter\,(\sigma_n(v, s, id).res, v) \wedge cap(v, e) > 0$$
$$\Rightarrow cap(v, e) > \sum_{\substack{\tilde{r} \in rng(resM_v):\\ resEg(\tilde{r}) = e}} allocBW(\tilde{r}.vrs, t)$$

We need to show that

$$resM'_v = filter\,(\sigma_{n+1}(v, s, id).res, v) \wedge cap(v, e) > 0$$
$$\Rightarrow cap(v, e) > \sum_{\substack{\tilde{r} \in rng\left(resM'_v\right):\\ resEg(\tilde{r}) = e}} allocBW(\tilde{r}.vrs, t)$$

The only reservation that changed from σ_n to σ_{n+1} is r to r', hence it holds (a)

$$\begin{aligned}
&\textstyle\sum_{\substack{\tilde{r}\in rng\left(resM'_v\right):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)\\
&= \textstyle\sum_{\substack{\tilde{r}\in rng\left(resM'_v\right):\\ \tilde{r}\neq r' resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})\\
&= \textstyle\sum_{\substack{\tilde{r}\in rng(resM_v):\\ \tilde{r}\neq r resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})
\end{aligned}$$

Furthermore, it holds that

$$(b)\;\; r'.vrs(\tilde{m}.idx).resBW = \min\left(\tilde{m}'.maxBW, \min(\tilde{m}'.accBW)\right)$$

and for all other indices $idx \neq \tilde{m}.idx$ it holds that

$$(c)\;\; r'.vrs(\tilde{m}.idx).resBW = r.vrs(\tilde{m}.idx).resBW$$

There are two cases:

$r'.vrs(\tilde{m}.idx).resBW \leq allocBW(r.vrs, \tilde{t})$: From this together with (b) and (c) it follows

$$(d)\; allocBW(r.vrs, \tilde{t}) = allocBW(r'.vrs, \tilde{t}).$$

From this it follows by (c) and IH

$$\begin{aligned}
&\textstyle\sum_{\substack{\tilde{r}\in rng\left(resM'_v\right):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)\\
&=^{(c)} \textstyle\sum_{\substack{\tilde{r}\in rng(resM_v):\\ \tilde{r}\neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})\\
&=^{(d)} \textstyle\sum_{\substack{\tilde{r}\in rng(resM_v):\\ \tilde{r}\neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r.vrs, \tilde{t})\\
&= \textstyle\sum_{\substack{\tilde{r}\in rng(resM_v):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)\\
&<^{IH} cap(v, e)
\end{aligned}$$

and therefore the claim.

$r'.vrs(\tilde{m}.idx).resBW > allocBW(r.vrs, \tilde{t})$: In this case by (b), (c), and the definition of $allocBW$ it follows

$$allocBW(r'.vrs, \tilde{t}) = r'.vrs(\tilde{m}.idx).resBW$$

and together with the definition of *finBW* it follows

$$\begin{aligned} & r'.vrs(\tilde{m}.idx).resBW \\ & \qquad := \min(\tilde{m}'.maxBW, \min(\tilde{m}'.accBW)) \\ & \qquad \leq \tilde{m}'.accBW[\tilde{m}'.ptr].avBW \\ & \qquad = avail(\tilde{m}, resM_v, \delta, \tilde{t}) \end{aligned}$$

hence, altogether it holds

$$(e)\ allocBW(r'.vrs, \tilde{t}) \leq avail(\tilde{m}, resM_v, \delta, \tilde{t})$$

By the definition of *avail*

$$\begin{aligned} & avail(m, resM, \delta, t) = \\ & \textbf{let} \\ & \ (\!|\ i, v, e\ |\!) = cur(m) \\ & \ resM' = resM\ ((v, src(m), m.id) \mapsto \bot) \\ & \ resM'_v = filter(resM', v) \\ & \textbf{in} \\ & \delta \cdot \left(cap(v,e) - \sum\nolimits_{\substack{r \in rng(resM'_v): \\ resEg(r)=e}} allocBW(r.vrs, t) \right) \end{aligned}$$

Applying this to (c) and

$$\begin{aligned} & \sum\nolimits_{\substack{\tilde{r} \in rng(resM'_v): \\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \\ & =^{(c)} \sum\nolimits_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t}) \\ & =^{(e)} \sum\nolimits_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + avail(\tilde{m}, resM_v, \delta, \tilde{t}) \\ & =^{(f)} \sum\nolimits_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + \\ & \qquad \delta \left(cap(v,e) - \sum\nolimits_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \right) \\ & = \delta cap(v,e) + (1-\delta) \sum\nolimits_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \\ & \leq \delta cap(v,e) + (1-\delta) \sum\nolimits_{\substack{\tilde{r} \in rng(resM_v): \\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \\ & <^{IH} \delta cap(v,e) + (1-\delta) cap(v,e) = cap(v,e) \end{aligned}$$

and the claim follows.

$TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: Analogous to CMP.

$UPT(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: The only relevant case is if $\tilde{v} = v$. Then by the event's guard *isRsvd*$(\sigma_n.res, \tilde{v}, \tilde{m}, \tilde{t})$ it follows

$$(11)\ r.vrs(\tilde{m}.idx).resBW \geq \min(\tilde{m}.accBW)$$

Then by the event's action it follows for the updated reservation $r' = \sigma_{n+1}.res(v, src(\tilde{m}), \tilde{m}.id)$ that

$$r'.vrs(\tilde{m}'.idx).resBW := \min(\tilde{m}'.maxBW, \min(\tilde{m}'.accBW))$$

and therefore

$$r'.vrs(\tilde{m}'.idx).resBW \leq r.vrs(\tilde{m}'.idx).resBW$$

From this and the fact that the version with index ($\tilde{m}'.idx$ is the only entry changed in the reservation map it follows

$$(g)\ \ allocBW(r'.vrs, \tilde{t}) \leq allocBW(r.vrs, \tilde{t})$$

From this together with the *IV* it follows as in event CMP that

$$\begin{aligned}
&\sum_{\substack{\tilde{r} \in rng(resM'_v):\\ resEg(\tilde{r}) = e}} allocBW(\tilde{r}.vrs, t)\\
&=^{(c)} \sum_{\substack{\tilde{r} \in rng(resM_v):\\ \tilde{r} \neq r \wedge resEg(\tilde{r}) = e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})\\
&\leq^{(g)} \sum_{\substack{\tilde{r} \in rng(resM_v):\\ \tilde{r} \neq r \wedge resEg(\tilde{r}) = e}} allocBW(\tilde{r}.vrs, t) + allocBW(r.vrs, \tilde{t})\\
&= \sum_{\substack{\tilde{r} \in rng(resM_v):\\ resEg(\tilde{r}) = e}} allocBW(\tilde{r}.vrs, t)\\
&<^{IH} cap(v, e)
\end{aligned}$$

and therefore the claim.

$BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: Analogous to FWD.

$FIN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: Analogous to FWD.

In case of the other events, reservations get removed, hence, *allocBW*$(r.vrs, t)$ stays the same or decreases for a corresponding reservation and the claim holds trivially.

□

B.10.2 Lower *ideal* Bound

Let m be a valid message with $(*)$ $m.minBW = 0$, $(+)$ $nodes(m) \subseteq H$, source s, ID id, and first AS v together with v's ingress and egress interface i_v and e_v. There is a strictly positive lower bound $G \cdot r_v$ on the *ideal* computation (even when all sources exceed their demands), where G only depends on m's path and $r_v = reqRatio(s, id, i_v, e_v)$ is the request ratio of m at v.

$$\begin{aligned}
&\forall evt \in \{CMP, TRN, UPT\}.\\
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N}.\\
&\lambda_n = evt(m, m', v, i, t) \Rightarrow\\
&\exists\, G \in]0;1].\ \forall x \in sgmt(m).\ ideal(m, resM_x) > G \cdot r_v
\end{aligned}$$

Lemma 53.

$$\begin{aligned}
&\forall m \in \mathcal{M}_R, s \in H, v \in sgmt(m) \setminus \{first(m)\}, i, e \in I, resM \in ResMap, t \in \mathbb{N}.\\
&cur(m) = v_i^e \ \wedge\ s = src(m) \ \wedge\ vrs = resM(v, src(m), m.id).vrs(m.idx) \wedge\\
&0 < vrs.maxBW = m.maxBW \wedge\\
&m.accBW[m.first].idBW = m.maxBW \wedge\\
&vrs.idBW = m.accBW[m.ptr].idBW \wedge\\
&egDem(v, src(m), e, resM, t) \leq cap(v, e) \wedge\\
&inDem(v, src(m), i, resM, t) \leq cap(v, i) \wedge\\
&transitDem(v, i, resM, t) \leq cap(v, i)\\
&\quad \Rightarrow \exists\, G_v \in]0;1].\ ideal(m, resM, \delta, t) \geq G_v \cdot m.accBW[m.ptr].idBW
\end{aligned}$$

and for $v = first(m)$

$$\exists\, G'_v \in]0;1].\ ideal(m, resM, \delta, t) \geq G'_v \cdot reqRatio_{start}(v, s, id, i, resM, t)$$

Proof. Given m, $v \in sgmt(m)$, $i, e \in I$, $resM \in ResMap$, and $t \in \mathbb{N}$ with $cur(m) = v_i^e$, $src(m) = s$, and

$$\begin{aligned}
(a)&\ \ 0 < m.maxBW\\
(b)&\ \ egDem(v, src(m), e, resM, t) \leq cap(v, e)\\
(c)&\ \ inDem(v, src(m), i, resM, t) \leq cap(v, i)\\
(d)&\ \ transitDem(v, i, resM, t) \leq cap(v, i)\\
(e)&\ \ 0 < vrs.maxBW = m.maxBW\\
(f)&\ \ vrs.idBW = m.accBW[m.ptr].idBW\\
(g)&\ \ m.accBW[m.first].idBW = m.maxBW
\end{aligned}$$

By the definition of *ideal*

$$reqRatio \cdot linkRatio \cdot tubeRatio(v,i,e,resM'_v,t) \cdot \delta \cdot cap(v,e)$$

We need to show that there are lower bounds for each of the following factors:

tubeRatio : By its definition

$$\begin{aligned} &tubeRatio(v,i,e,resM,t) \\ &= \frac{\min\{cap(v,i), tubeDem(v,i,e,resM,t)\}}{\sum_{i' \in I} \min\{cap(v,i'), tubeDem(v,i',e,resM,t)\}} \end{aligned}$$

First we derive a lower bound for the fraction's nominator. By the definition of *tubeDem*

$$\begin{aligned} &tubeDem(v,i,e,resM,t) \\ &= \sum_{\substack{r \in rng(resM): \\ resIn(r)=i \\ resEg(r)=e}} adjReqDem(v,r,i,e,resM,t) \end{aligned}$$

A lower bound is the summand $r = resM(v,s,m.id)$

$$\begin{aligned} &adjReqDem(v,r,resM,t) = \\ &= \min\{inScalFctr(v,s,i,resM,t), egScalFctr(v,s,e,resM,t)\} \\ &\qquad \cdot reqDem(v,r,i,e,t) \end{aligned}$$

From (b) and the definition of *egScalFctr*

$$\begin{aligned} &egScalFctr(v,s,e,resM,t) = \\ &\frac{\min(cap(v,e), egDem(v,s,e,resM,t))}{egDem(v,s,e,resM,t)}. \end{aligned}$$

it follows that

$$egScalFctr(v,s,e,resM,t) = 1$$

and the same for *inScalFctr* by (c). Hence, it is sufficient to provide a lower bound for *reqDem*

$$\begin{aligned} &reqDem(v,r,i,e,t) \\ &= \min\{cap(v,i), cap(v,e), demBW(r.vrs,t)\} \end{aligned}$$

By the definition of *egDem*

$$egDem(v,s,e,resM,t) = \\ \sum_{\substack{r' \in rng(resM): \\ resSr(r')=s \\ resEg(r')=e}} reqDem(v,r',resIn(r'),e,t).r.path[r.ptr].inI,e).$$

and (b) it follows that

$$(A) \;\; reqDem(v,r,i,e,t) \leq egDem(v,s,e,resM,t) \leq^{(b)} cap(v,e)$$

and analogously

$$reqDem(v,r,i,e,t) \leq inDem(v,s,i,resM,t) \leq^{(c)} cap(v,i)$$

it follows that

$$reqDem(v,r,i,e,t) = demBW(r.vrs,t).$$

By the definition of *demBW*

$$demBW(vrsM,t) = \\ \max_{\substack{vrs\in \\ rng(vrsM)}} \{vrs.maxBW \mid vrs.minBW \leq vrs.resBW \wedge vrs.expT \geq t\}$$

it follows that

$$(B) \;\; demBW(vrsM,t) \geq r.vrs(m.idx).maxBW = m.maxBW$$

All together we obtain that

$$\begin{aligned} &tubeRatio(v,i,e,resM,t) \\ &= \frac{\min\{cap(v,i), tubeDem(v,i,e,resM,t)\}}{\sum_{i'\in I} \min\{cap(v,i'), tubeDem(v,i',e,resM,t)\}} \\ &\geq \frac{\min\{cap(v,i), tubeDem(v,i,e,resM,t)\}}{\sum_{i'\in I_x} cap(v,i')} \\ &\geq^{(A),(B)} \frac{m.maxBW}{\sum_{i'\in I_v} cap(v,i')} \end{aligned}$$

with $I_v := \{i' \in I \mid cap(v,i') > 0\}$.

linkRatio There are two cases $v = \mathit{first}(m)$ and $v \in \mathit{sgmt}(m) \setminus \mathit{first}(m)$. By the definition of $\mathit{linkRatio}_{\mathit{transit}}$

$$\begin{aligned}
&\mathit{linkRatio}_{\mathit{transit}}(v, i, \mathit{resM}, t) = \\
&\textbf{let} \\
&\quad \mathit{stDem} = \mathit{startDem}(v, i, \mathit{resM}, t) \\
&\quad \mathit{trDem} = \mathit{transitDem}(v, i, \mathit{resM}, t) \\
&\textbf{in} \\
&\quad \frac{\min\{\mathit{cap}(v,i), \mathit{trDem}\}}{\min\{\mathit{cap}(v,i), \mathit{stDem}\} + \min\{\mathit{cap}(v,i), \mathit{trDem}\}}.
\end{aligned}$$

By (d) it follows for its nominator

$$\begin{aligned}
(D)\ &\min\{\mathit{cap}(v,i), \mathit{trDem}\} \\
&= \min\{\mathit{cap}(v,i), \mathit{transitDem}(v, i, \mathit{resM}, t)\} \\
&=^{(d)} \mathit{transitDem}(v, i, \mathit{resM}, t)
\end{aligned}$$

and therefore

$$\begin{aligned}
&\mathit{linkRatio}_{\mathit{transit}}(v, i, \mathit{resM}, t) \\
&= \frac{\min\{\mathit{cap}(v,i), \mathit{trDem}\}}{\min\{\mathit{cap}(v,i), \mathit{stDem}\} + \min\{\mathit{cap}(v,i), \mathit{trDem}\}} \\
&\geq \frac{\min\{\mathit{cap}(v,i), \mathit{trDem}\}}{2 \cdot \mathit{cap}(v,i)} \\
&\geq^{(D)} \frac{\mathit{transitDem}(v, i, \mathit{resM}, t)}{2 \cdot \mathit{cap}(v,i)}
\end{aligned}$$

$\mathit{linkRatio}_{\mathit{start}}$: In this case (D) does not hold, but by By (C) and (g) it follows

$$\begin{aligned}
(E)\ &\mathit{adjIdDem}(v, r, \mathit{resM}, t) \\
&\geq^{(C)} m.\mathit{accBW}[m.\mathit{ptr}].\mathit{idBW} \\
&=^{(g)} m.\mathit{maxBW}
\end{aligned}$$

and we obtain as lower bound

$$\begin{aligned}
&\mathit{linkRatio}_{\mathit{start}}(v,i,\mathit{resM},t)\\
&= \frac{\min\{\mathit{cap}(v,i),\mathit{stDem}\}}{\min\{\mathit{cap}(v,i),\mathit{stDem}\}+\min\{\mathit{cap}(v,i),\mathit{trDem}\}}\\
&\geq \frac{\min\{\mathit{cap}(v,i),\mathit{stDem}\}}{2\cdot \mathit{cap}(v,i)}\\
&\geq^{(E)} \frac{m.\mathit{maxBW}}{2\cdot \mathit{cap}(v,i)}
\end{aligned}$$

$\mathit{reqRatio}_{\mathit{transit}}$: By the definition of $\mathit{adjIdDem}$

$$\begin{aligned}
&\mathit{adjIdDem}(v,r,\mathit{resM},t) =\\
&\ \mathit{egScalFctr}(v,s,e,\mathit{resM},t)\cdot \min\{\mathit{cap}(v,i),\mathit{cap}(v,e),\mathit{idBW}(r.\mathit{vrs},t)\}.
\end{aligned}$$

and similar as above $\mathit{egScalFctr}(v,s,e,\mathit{resM},t) = 1$ it follows that

$$\begin{aligned}
(C)\ &\mathit{adjIdDem}(v,r,\mathit{resM},t)\\
&= \mathit{idBW}(r.\mathit{vrs},t)\\
&\geq r.\mathit{vrs}(m.\mathit{idx}).\mathit{idBW}\\
&= m.\mathit{accBW}[m.\mathit{ptr}].\mathit{idBW}
\end{aligned}$$

By (C) and (d) and the definition of $\mathit{reqRatio}_{\mathit{transit}}$

$$\mathit{reqRatio}_{\mathit{transit}}(v,s,id,i,\mathit{resM},t) = \frac{\mathit{adjIdDem}(v,\mathit{resM}(v,s,id),\mathit{resM},t)}{\mathit{transitDem}(v,i,\mathit{resM},t)}$$

it follows

$$\begin{aligned}
&\mathit{reqRatio}_{\mathit{transit}}(v,s,id,i,\mathit{resM},t)\\
&= \frac{\mathit{adjIdDem}(v,\mathit{resM}(v,s,id),\mathit{resM},t)}{\mathit{transitDem}(v,i,\mathit{resM},t)}\\
&\geq^{(C)} \frac{m.\mathit{accBW}[m.\mathit{ptr}].\mathit{idBW}}{\mathit{transitDem}(v,i,\mathit{resM},t)}
\end{aligned}$$

$\mathit{reqRatio}_{\mathit{start}}$: By (E) and the definition of $mathitreqRatio_{\mathit{start}}$ it follows

$$\begin{aligned}
&\mathit{reqRatio}_{\mathit{start}}(v,s,id,i,\mathit{resM},t)\\
&= \frac{\mathit{adjIdDem}(v,\mathit{resM}(v,s,id),\mathit{resM},t)}{\mathit{startDem}(v,i,\mathit{resM},t)}\\
&\geq^{(E)} \frac{m.\mathit{maxBW}}{\mathit{startDem}(v,i,\mathit{resM},t)}
\end{aligned}$$

Altogether we obtain the following lower bounds for *ideal* depending on the two cases $v = \mathit{first}(m)$ and $v \in \mathit{sgmt}(m) \setminus \{\mathit{first}(m)\}$, respectively:

$v = \mathit{first}(m)$:

$$\begin{aligned}
&\mathit{ideal}(m, \mathit{resM}, \delta, t) \\
&= \mathit{reqRatio}_{\mathit{start}} \cdot \mathit{linkRatio}_{\mathit{start}} \cdot \mathit{tubeRatio} \cdot \delta \cdot \mathit{cap}(v, e) \\
&\geq \mathit{reqRatio}_{\mathit{start}} \cdot \frac{m.\mathit{maxBW}}{2 \cdot \mathit{cap}(v, i)} \cdot \frac{m.\mathit{maxBW}}{\sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e) \\
&= \mathit{reqRatio}_{\mathit{start}} \cdot \frac{m.\mathit{maxBW}^2}{2 \cdot \mathit{cap}(v, i) \cdot \sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e) \\
&\geq \frac{m.\mathit{maxBW}}{\mathit{startDem}(v, i, \mathit{resM}, t)} \cdot \frac{m.\mathit{maxBW}^2}{2 \cdot \mathit{cap}(v, i) \cdot \sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e)
\end{aligned}$$

Hence, we can set

$$G'_v := \frac{m.\mathit{maxBW}^2}{2 \cdot \mathit{cap}(v, i) \cdot \sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e)$$

which only depends on *m.maxBW* and the capacities on *m.path* of the network.

$v \in \mathit{sgmt}(m) \setminus \{\mathit{first}(m)\}$:

$$\begin{aligned}
&\mathit{ideal}(m, \mathit{resM}, \delta, t) \\
&= \mathit{reqRatio}_{\mathit{transit}} \cdot \mathit{linkRatio}_{\mathit{transit}} \cdot \mathit{tubeRatio} \cdot \delta \cdot \mathit{cap}(v, e) \\
&\geq \frac{m.\mathit{accBW}[m.\mathit{ptr}].\mathit{idBW}}{\mathit{trDem}} \cdot \frac{\mathit{trDem}}{2 \cdot \mathit{cap}(v, i)} \cdot \frac{m.\mathit{maxBW}}{\sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e) \\
&= m.\mathit{accBW}[m.\mathit{ptr}].\mathit{idBW} \cdot \frac{m.\mathit{maxBW}}{2 \cdot \mathit{cap}(v, i) \cdot \sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e)
\end{aligned}$$

Hence, we can set

$$G_v := \frac{m.\mathit{maxBW}}{2 \cdot \mathit{cap}(v, i) \cdot \sum_{i' \in I_v} \mathit{cap}(v, i')} \cdot \delta \cdot \mathit{cap}(v, e)$$

which only depends on *m.maxBW* and the capacities on *m.path* of the network.

□

B.10.3 Bounded Tube-Proportionality

Provided that two ingress links i, i' of AS x are not congested, the *tubeRatio* computation splits the capacity of the egress link e proportionally according to the tube demands of i and i' to e

$$\frac{tubeRatio(i,e)}{tubeRatio(i',e)} = \frac{tubeDem(i,e)}{tubeDem(i',e)}.$$

In case i' is congested and its tube demand to e further increases, the ratio between both tube ratios remains fixed

$$\frac{tubeRatio(i,e)}{tubeRatio(i',e)} = \frac{tubeDem(i,e)}{cap(x,i')}.$$

Proof. Given two ingress links i, i' of AS x. If both ingress links i and i' are not congested, i.e.,

$$\begin{array}{ll} (a) & \sum_{\tilde{s}\in V} inDem(x,\tilde{s},i,resM,t) \le cap(x,i) \\ (b) & \sum_{\tilde{s}\in V} inDem(x,\tilde{s},i',resM,t) \le cap(x,i'). \end{array}$$

By this it follows and the definition of *tubeDem* it follows

$$\begin{aligned} (a')\ & tubeDem(v,i,e,resM,t) \\ &= \sum_{\substack{r\in rng(resM):\\ resIn(r)=i\wedge resEg(r)=e}} adjReqDem(v,r,i,e,resM,t) \\ &\le \sum_{\substack{r\in rng(resM):\\ resIn(r)=i\wedge resEg(r)=e}} reqDem(v,r,i,e,resM,t) \\ &\le \sum_{\substack{r\in rng(resM):\\ resIn(r)=i}} reqDem(v,r,i,e,resM,t) \\ &= \sum_{\tilde{s}\in V} \sum_{\substack{r\in rng(resM):\\ resSr(r)=\tilde{s}\wedge resIn(r)=i}} reqDem(v,r,i,resEg(r),t) \\ &= \sum_{\tilde{s}\in V} inDem(x,\tilde{s},i,resM,t) \\ &\le^{(a)} cap(x,i) \end{aligned}$$

and similarly

$$(b')\ tubeDem(v,i',e,resM,t) \le cap(x,i')$$

therefore it follows

$$\frac{tubeRatio(x,i,e,resM,t)}{tubeRatio(x,i',e,resM,t)} = \frac{\frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}}{\frac{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}} = \frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}} =^{(a'),(b')} \frac{tubeDem(x,i,e,resM,t)}{tubeDem(x,i',e,resM,t)}$$

Independent from ingress link i' being congestion, if i is not congested then it follows

$$\frac{tubeRatio(x,i,e,resM,t)}{tubeRatio(x,i',e,resM,t)} = \frac{\frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}}{\frac{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}} = \frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}} \geq^{(a')} \frac{tubeDem(x,i,e,resM,t)}{cap(x,i')}.$$

If the tube demand between i' and e exceeds $cap(x,i')$, i.e.,

$$tubeDem(x,i',e,resM,t) \geq cap(x,i')$$

then the last inequality becomes an equality

$$\frac{tubeRatio(x,i,e,resM,t)}{tubeRatio(x,i',e,resM,t)} = \frac{tubeDem(x,i,e,resM,t)}{cap(x,i')}$$

and stays fixed no matter how much $tubeDem(x,i',e,resM,t)$ increases. □

B.10.4 Per Request-Proportionality

Suppose two sources s and s' respectively make new reservations m and m' whose paths intersect on a connected segment $[v_1,\dots,v_n]$, i.e.,

$$\exists k_1,k_n,k'_1,k'_n \in \mathbb{N}.k_1 \leq k_n \leq length(m.path) \land k'_1 \leq k'_n \leq length(m'.path) \land$$
$$\forall k_i,k'_i \in \mathbb{N}.\ k_1 \leq k_i \leq k_n, k'_1 \leq k'_i \leq k'_n \Rightarrow$$
$$v_i = m.path[k_i].as = m.path[k'_i].as$$

If s and s' do not have excessive demands on this segment, then the ratio of their *ideal* bandwidth computations on the segment remain the same to their ratio at the first AS on the segment, even if links on the segment are congested

$$\begin{aligned}
&\forall v_i \in \{v_1, \dots, v_n\}. \\
&\frac{ideal(m, resM_{v_i}, \delta, t)}{ideal(m', resM_{v_i}, \delta, t)} = \frac{ideal(m, resM_{v_1}, \delta, t)}{ideal(m', resM_{v_1}, \delta, t)}.
\end{aligned}$$

Lemma 54.

$$\begin{aligned}
&\forall m, m' \in \mathcal{M}_R, s, s' \in V, v \in H, i, e \in I, resM \in ResMap, t \in \mathbb{N} \\
&m.path[m.ptr] = m'.path[m'.ptr] = v_i^e \;\wedge \\
&s = src(m) \;\wedge\; s' = src(m') \;\wedge \\
&inDem(v, s, i, resM_v, t) \le cap(v, i) \;\wedge \\
&egDem(v, s', e, resM_v, t) \le cap(v, e) \\
&\Rightarrow \frac{ideal(m, resM, \delta, t)}{ideal(m', resM, \delta, t)} = \frac{resM(v, s, m.id).vrs(m.idx).idBW}{resM(v, s', m'.id).vrs(m'.idx).idBW}
\end{aligned}$$

Proof. Given $m, m' \in \mathcal{M}_R$, $s, s' \in V$, $v \in H$, $i, e \in I$, $resM \in ResMap$, $t \in \mathbb{N}$ with

$$\begin{aligned}
&m.path[m.ptr] = m'.path[m'.ptr] = v_i^e \\
&s = src(m), s' = src(m')
\end{aligned}$$

and that s and s' have modest demands, i.e.,

$$\begin{aligned}
&(a)\;\; inDem(v, s, i, resM_v, t), inDem(v, s', i, resM_v, t) \le cap(v, i) \\
&(b)\;\; egDem(v, s, e, resM_v, t), egDem(v, s', e, resM_v, t) \le cap(v, e).
\end{aligned}$$

By this it follows that both *inScalFctr* and *egScalFctr* are equal to 1 and therefore $adjReqDem = reqDem$ for m and m', respectively. By the definition of *tubeRatio*

$$\begin{aligned}
&ideal(m, resM, \delta, t)\\
&= reqRatio_{transit} \cdot linkRatio_{transit} \cdot tubeRatio \cdot \delta \cdot cap(v,e)\\
&= \frac{adjIdDem(v,r,resM,t)}{trDem} \cdot \frac{\min\{cap(v,i), trDem\}}{\min\{cap(v,i), stDem\} + \min\{cap(v,i), trDem\}}\\
&\cdot \frac{\min\{cap(v,i), tubeDem(v,i,e,resM,t)\}}{\sum_{i' \in I} \min\{cap(v,i'), tubeDem(v,i',e,resM,t)\}} \cdot \delta \cdot cap(v,e)\\
&= \frac{adjIdDem(v,r,resM,t)}{stDem + trDem} \cdot \frac{tubeDem(v,i,e,resM,t)}{\sum_{i' \in I} \min\{cap(v,i'), tubeDem(v,i',e,resM,t)\}}\\
&\quad \cdot \delta \cdot cap(v,e)
\end{aligned}$$

□

Bibliography

[1] BBC - Business: The Economy, how Leeson broke the bank. `http://news.bbc.co.uk/2/hi/business/375259.stm`, 1999. [Accessed: 23-May-2019].

[2] Verizon - 2019 Data Breach Investigations Report: Incident Classification Patterns and Subsets. `https://enterprise.verizon.com/resources/reports/dbir/2019/incident-classification-patterns-subsets/`, 2019. [Accessed: 28-May-2019].

[3] Ali E Abdallah and Etienne J Khayat. A formal model for parameterized role-based access control. In *Formal Aspects in Security and Trust*, pages 233–246. Springer, 2005.

[4] Gail-Joon Ahn and Ravi Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 43–54. ACM, 1999.

[5] Mohammad A Al-Kahtani and Ravi Sandhu. A model for attribute-based user-role assignment. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 353–362. IEEE, 2002.

[6] Mohammad A Al-Kahtani and Ravi Sandhu. Induced role hierarchies with attribute-based RBAC. In *Proceedings of the eighth ACM symposium on access control models and technologies*, pages 142–148. ACM, 2003.

[7] Konstantine Arkoudas, Ritu Chadha, and Jason Chiang. Sophisticated access control via SMT and logical frameworks. *ACM Transactions on Information and System Security (TISSEC)*, 16(4):17, 2014.

[8] Alessandro Armando and Silvio Ranise. Automated and efficient analysis of role-based access control with attributes. In *Data and Applications Security and Privacy XXVI*, pages 25–40. Springer, 2012.

[9] Ezedin Barka and Ravi Sandhu. A role-based delegation model and some extensions. In *23rd National Information Systems Security Conference*, pages 396–404. Citeseer, 2000.

[10] Cristina Basescu, Raphael M Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei

Urakawa. Sibra: Scalable internet bandwidth reservation architecture. *arXiv preprint arXiv:1510.02696*, 2015.

[11] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. Automated analysis of security-design models. *Information and Software Technology*, 51(5):815–831, 2009.

[12] Moritz Y Becker, Cédric Fournet, and Andrew D Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[13] Moritz Y Becker and Peter Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*, pages 139–154. IEEE, 2004.

[14] Rafae Bhatti, Arif Ghafoor, Elisa Bertino, and James BD Joshi. X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security (TISSEC)*, 8(2):187–227, 2005.

[15] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC 2475 (Informational), December 1998. Updated by RFC 3260.

[16] Piero Bonatti, Clemente Galdi, and Davide Torres. ERBAC: event-driven RBAC. In *Proceedings of the 18th ACM symposium on Access control models and technologies*, pages 125–136. ACM, 2013.

[17] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[18] Bob Briscoe. Flow rate fairness: Dismantling a religion. *SIGCOMM Comput. Commun. Rev.*, 37(2):63–74, March 2007.

[19] Carlos Cotrini, Thilo Weghorn, David Basin, and Manuel Clavel. Analyzing first-order role based access control. In *Proceedings of the 2015 IEEE 28th Computer Security Foundations Symposium*, CSF '15, pages 3–17, Washington, DC, USA, 2015. IEEE Computer Society.

[20] Samantha Dalton. Kweku Adoboli: From 'rising star' to rogue trader. `https://web.archive.org/web/20130110185312/http://www.bbc.co.uk/news/uk-19660659`, 2012. [Accessed: 23-May-2019].

[21] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[22] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Comp. Comm. Rev.*, 1989.

[23] Dietrich G. Aumann. Twelve Nights. unpublished manuscript, 2019.

[24] Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *Automated Reasoning*, pages 632–646. Springer, 2006.

[25] Adrian Farrel, Jean-Philippe Vasseur, and Jerry Ash. A path computation element (PCE)-based architecture. Technical report, 2006.

[26] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[27] Anna Lisa Ferrara, P Madhusudan, and Gennaro Parlato. Policy analysis for self-administrated role-based access control. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 432–447. Springer, 2013.

[28] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 196–205, New York, NY, USA, 2005. ACM.

[29] Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *Proceedings of the second ACM workshop on Role-based access control*, pages 153–159. ACM, 1997.

[30] P Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In *ACM SIGCOMM Comp. Comm. Rev.*, 2009.

[31] Joseph Y Halpern and Vicky Weissman. Using first-order logic to reason about policies. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):21, 2008.

[32] Garrett Hardin. The tragedy of the commons. *Science*, 1968.

[33] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Soo Bum Lee, Xin Zhang, Sangjae Yoo, Virgil Gligor, and Adrian Perrig. STRIDE: Sanctuary trail – refuge from internet DDoS entrapment. In *AsiaCCS*, 2013.

[34] Jingwei Huang, David M Nicol, Rakesh Bobba, and Jun Ho Huh. A framework integrating attribute-based policies into role-based access control. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 187–196. ACM, 2012.

[35] F. Baker J. Babiarz, K. Chan. Configuration Guidelines for DiffServ Service Classes.

[36] Rahul Jain and Jean Walrand. An efficient Nash-implementation mechanism for network resource allocation. *Automatica*, 46(8):1276–1283, 2010.

[37] Somesh Jha, Ninghui Li, Mahesh Tripunitara, Qihua Wang, and William H Winsborough. Towards formal verification of role-based access control policies. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):242–255, 2008.

[38] Xin Jin, Ram Krishnan, and Ravi Sandhu. A role-based administration model for attributes. In *Proceedings of the First International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12. ACM, 2012.

[39] Xin Jin, Ram Krishnan, and Ravi Sandhu. Reachability analysis for role-based administration of attributes. In *Proceedings of the 2013 ACM workshop on Digital identity management*, pages 73–84. ACM, 2013.

[40] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: role-centric attribute-based access control. In *Computer Network Security*, pages 84–96. Springer, 2012.

[41] Ramesh Johari and John N Tsitsiklis. Efficiency of scalar-parameterized mechanisms. *Operations Research*, 57(4):823–839, 2009.

[42] James BD Joshi. Access-control language for multidomain environments. *Internet Computing, IEEE*, 8(6):40–50, 2004.

[43] Min Suk Kang and Virgil D. Gligor. Routing bottlenecks in the internet: Causes, exploits, and countermeasures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 321–333, New York, NY, USA, 2014. ACM.

[44] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. The Crossfire Attack. In *IEEE S&P*, 2013.

[45] Frank P Kelly, Aman K Maulloo, and David KH Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 49(3):237–252, 1998.

[46] Vladimir Kolovski, James Hendler, and Bijan Parsia. Analyzing web access control policies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 677–686, New York, NY, USA, 2007. ACM.

[47] Shriram Krishnamurthi. The CONTINUE server (or, How I administered PADL 2002 and 2003). In *Practical aspects of declarative languages*, pages 2–16. Springer, 2003.

[48] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[49] Soo Bum Lee and Virgil D. Gligor. FLoc: Dependable link access for legitimate traffic in flooding attacks. In *IEEE ICDCS*, 2010.

[50] Soo Bum Lee, Min Suk Kang, and Virgil D. Gligor. CoDef: Collaborative defense against large-scale link-flooding attacks. In *ACM CoNEXT*, 2013.

[51] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480 (Informational), February 2012.

[52] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML 2002 The Unified Modeling Language*, pages 426–441. Springer, 2002.

[53] J. Nagle. On Packet Switches With Infinite Storage. RFC 970, December 1985.

[54] Tim Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, Shriram Krishnamurthi, and Varun Singh. The Margrave tool. http://www.margrave-tool.org/v3/. Accessed: 2015-01-12.

[55] Timothy Nelson, Christopher Barratt, Daniel J Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The Margrave tool for firewall analysis. In *LISA*, 2010.

[56] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *ACM SIGCOMM*, 2007.

[57] Adrian Perrig, Pawel Szalachowski, Raphael M Reischuk, and Laurent Chuat. *SCION: a secure internet architecture*. Springer, 2017.

[58] Salman Saghafi, Tim Nelson, and Daniel J Dougherty. Geometric logic for policy analysis. In *International Workshop on Automated Reasoning in Security and Software Verification (ARSEC 2013)*, pages 12–20, 2013.

[59] Ravi Sandhu. Role hierarchies and constraints for lattice-based access controls. In *Computer Security ESORICS 96*, pages 65–79. Springer, 1996.

[60] Stanislav Shalunov and Benjamin Teitelbaum. Quality of service and denial of service. In *Proceedings of the ACM SIGCOMM Workshop on Revisiting IP QoS: What Have We Learned, Why Do We Care?*, RIPQoS '03, pages 137–140, New York, NY, USA, 2003. ACM.

[61] Rayadurgam Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[62] Scott D Stoller, Ping Yang, Mikhail I Gofman, and CR Ramakrishnan. Symbolic reachability analysis for parameterized administrative role-based access control. *Computers & Security*, 30(2):148–164, 2011.

[63] Ahren Studer and Adrian Perrig. The Coremelt attack. In *ESORICS*, 2009.

[64] Iain Thomson. Arbor networks hit with 1.7 Tbps DDoS attack. `https://web.archive.org/web/20180828011250/https://www.theregister.co.uk/2018/03/05/worlds_biggest_ddos_attack_record_broken_after_just_five_days/`, 2018. [Accessed: 23-May-2019].

[65] Iain Thomson. Gits club GitHub code tub with record-breaking 1.35 Tbps DDoS drub. https://web.archive.org/web/20180812154323/https://www.theregister.co.uk/2018/03/01/github_ddos_biggest_ever/, 2018. [Accessed: 23-May-2019].

[66] US-CERT. Alert (TA17-164A) HIDDEN COBRA – North Korea's DDoS Botnet Infrastructure. https://www.us-cert.gov/ncas/alerts/TA17-164A, 2017.

[67] JP. Vasseur and JL. Le Roux. Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440 (Proposed Standard), March 2009. Updated by RFC 7896.

[68] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[69] Abraham Yaar, Adrian Perrig, and Dawn Song. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE S&P*, 2004.

[70] Xiaowei Yang, David Clark, and Arthur W Berger. Nira: A new inter-domain routing architecture. *IEEE/ACM Transactions on Networking*, 2007.

[71] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-limiting network architecture. *ACM SIGCOMM Comp. Comm. Rev.*, 2005.

[72] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network*, 1993.

Resume

THILO LOTHAR PETER WEGHORN

Born on October 15, 1985 in Bamberg, Germany.
Citizen of Germany.

Education

2006 – 2013	**Diploma in Computer Science** LMU Munich, Germany
2005 – 2012	**Diploma in Mathematics** LMU Munich, Germany
2013 – 2019	**PhD in Computer Science** ETH Zurich, Switzerland