N-Tube: Formally Verified Secure Bandwidth Reservation in Path-Aware Internet Architectures

Anonymous Author(s)

ABSTRACT

We present N-Tube, a novel, provably secure, inter-domain bandwidth reservation algorithm that runs on a network architecture supporting path-based forwarding. N-Tube reserves global end-to-end bandwidth along network paths in a distributed, *neighbor-based*, and *tube-fair* way. It guarantees that benign bandwidth demands are granted *available* allocations that are *immutable*, *stable*, *lower-bounded*, and *fair*, even during adversarial demand bursts.

We formalize N-Tube and powerful adversaries as a labeled transition system, and inductively prove its safety and security properties. We additionally apply statistical model checking to validate our proofs and provide an additional quantitative assessment of N-Tube's guarantees. This provides strong guarantees for protection against DDoS attacks. We are not aware of any other complex networked system designs that have been subjected to a comparable analysis of both their qualitative properties (such as correctness and security) *and* their quantitative properties (such as performance).

NOTE. This is the full-version technical report accompanying our paper submission to ACM CCS 2021.

1 INTRODUCTION

Secure Bandwidth Allocation. Providing useful guarantees during DDoS attacks remains an open problem. The increasing sophistication of attacks has not yet been countered by progress in scalable, cost-effective defenses. Sophisticated attacks do not target the victim directly, but only a few critical network links carrying 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 [42]. Similarly, in Coremelt, an adversary sets up traffic flows among pairs of bots that it controls in a way that floods critical links [67]. In these strongest known attacks, an attacker with limited resources can effectively attack critical links and degrade connectivity for large Internet regions [41]. Current techniques cannot defend against such attacks since the congestion hotspots are outside the victims' control.

DDoS protection can be realized by an effective quality of service (QoS) scheme that provides hard bandwidth guarantees in the face of sophisticated adversaries. Since best-effort delivery and over-provisioned network bandwidth enable good performance in the average case, offering QoS guarantees requires fair resource allocation even when bandwidth becomes scarce [63]. Previous QoS architectures, such as IntServ [17] and DiffServ [15], were designed for the Internet with trusted network participants, not for adversarial scenarios. It remains an open research problem *how to allocate bandwidth in malicious contexts 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 [69]. Therefore, commonly known fairness notions that QoS solutions try to achieve, such as per source [56], per destination [78], per flow [26], per computation [58], and per class [38], are insufficient in such settings and result in unfair bandwidth allocations. These fairness notions suffer from the "tragedy of the commons" [36], 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, congestion-control-based 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].

New Internet architectures supporting *path-based forwarding* provide a means to mitigate the above-mentioned issues. Instead of using frequently updated forwarding tables as in today's Internet, they leverage path-based forwarding where the paths taken by data packets stay fixed and correspond to the reserved paths. This simplifies reasoning about resource allocation. Segment Routing [30], SCION [59], Pathlets [34], and NIRA [77] are prominent examples of such architectures, where the first two already see real-world deployment.

N-Tube Algorithm. To address the above challenges we present *N-Tube*, a new *Neighbor-based*, *Tube-fair* bandwidth reservation algorithm, along with a novel notion of allocation fairness called *bounded tube fairness*. N-Tube is designed for networks that support path-based forwarding and prevents in-network link congestion attacks, including the strongest known attacks like Coremelt/Crossfire. It is thus also robust against standard link-flooding attacks, including amplification attacks. To allocate bandwidth on a path, each on-path AS computes and allocates bandwidth locally while accounting for other reservations.

N-Tube builds on two key ideas. First, to always enable the allocation of some (non-zero) bandwidth, N-Tube only uses a fixed fraction of the available bandwidth, saving the rest for future reservation requests. By guaranteeing that the reserved bandwidth stays unchanged until expiry, N-Tube also enables a predictable stabilization period of the bandwidth allocations during times of stable bandwidth demands.

Second, with *bounded tube fairness*, each AS's aggregated bandwidth demands are first *bounded* by the available bandwidth, and then split proportionally among its immediate network neighbors. Hence, if a malicious AS (outside the honest path) tries to congest a link, the first honest AS between the attacker and targeted link limits the adversarial demands, thereby preventing it from obtaining a disproportional share of bandwidth on that link. Consequently, N-Tube also guarantees any honest source AS a lower bound on the allocated bandwidth, independently of the desired destination.

Verification Approach. Inter-domain bandwidth reservation is, in general, a difficult problem with complex bandwidth allocation dynamics especially for operation in adversarial environments. This necessitates the verification of any proposed bandwidth reservation algorithm to validate its intended properties, in particular to establish both qualitative correctness and security guarantees as well as quantitative guarantees about the system's bandwidth allocation dynamics. The verification of N-Tube's qualitative and quantitative properties is particularly challenging for several reasons: its desired properties must hold in the presence of a powerful adversary and for arbitrary network topologies. Moreover, the model involves unbounded state information and the verification requires non-linear arithmetical reasoning about bandwidth allocation. These features are notoriously hard to handle for automated verification tools.

We tackle this problem by using a combination of mathematical proofs for the qualitative properties and statistical verification and estimation for the quantitative properties. For qualitative guarantees, we verify N-Tube's correctness and security by: (i) formalizing the algorithm, together with the network environment and attackers, as a *labeled transition system* (LTS), (ii) specifying the safety and security properties as predicates over LTS executions, and (iii) proving by *induction* using careful pencil-and-paper proofs that the formal model satisfies these properties.

For quantitative guarantees, we analyze N-Tube's stability and fairness properties using statistical model checking (SMC) [62]. SMC has been successfully used to analyze large-scale distributed systems and has demonstrated its predictive power in an early design stage, i.e., its estimations are consistent with implementation-based evaluations under realistic deployment [16, 52]. SMC samples and analyzes system executions until a given confidence level is reached. Concretely, we use an independent formal framework, Maude [23], and its SMC tool [62]. We first express the LTS model as a *rewrite theory* executable in Maude, and then transform it into a *probabilistic rewrite theory* for the SMC-based analysis of N-Tube's quantitative properties. Unlike in implementation-based evaluations, this allows us to explore the large parameter space, to consider various (malicious) scenarios, and to obtain statistics with desired confidence level and error margins. With our SMC analysis we also obtain additional confidence in our inductive proofs of N-Tube's qualitative properties.

In networking, formal methods have been applied to verify qualitative and quantitative properties of network configurations, routing protocols, and DoS protection mechanisms. We will discuss this and additional related work in Section 7. However, we are not aware of any prior work that formally models and verifies a bandwidth reservation system, neither in benign nor in adversarial settings.

Main Contributions. We provide (i) the first principled solution to the global inter-domain bandwidth allocation problem that offers stable, lower-bounded, and fair bandwidth allocation in adversarial settings (Sections 3 and 4); (ii) the formalization of N-Tube, a strong attacker model, and all its safety and security properties, as well as inductive proofs establishing these properties (Sections 4.4 and 5); (iii) the automated statistical verification and estimation of N-Tube's behaviors, both to validate our proofs and to provide quantitative guarantees and assess N-Tube's resistance to attacks in various malicious scenarios (Section 6).

2 PRELIMINARIES

2.1 Design Goal and 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 is a *bandwidth reservation mechanism* that allocates bandwidth according to the demands of source ASes and guarantees a minimum bandwidth allocation even under heavy congestion or flooding attacks. Thus, N-Tube should satisfy the following properties:

- G1 **Availability**: Any successful reservation request can reserve bandwidth, in spite of network congestion.
- G2 **Immutability**: The allocated bandwidth of any existing reservation stays fixed until it expires.
- G3 **Stability**: In periods of steady and constant demand, the bandwidth allocation in the entire network stabilizes in a predictable period of time.
- G4 **Minimum Bandwidth Guarantee**: After the network stabilizes, there is a lower bound on the allocated bandwidth, i.e., a minimal bandwidth guarantee even with high external demands such as link-flooding attacks.
- G5 **Bounded Tube Fairness**: Bandwidth allocation is distributed proportionally to the requested demands, however, adjusted to the maximally available bandwidth.

Additional requirements ensure that N-Tube is efficient and practical from an operational perspective, see Appendix A.

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 denote physical links. Each link starts at an egress interface of an AS, called an *egress link* of the AS, and ends at an ingress interface of another AS, called an *ingress link* of that AS. Each edge has a weight that corresponds to the link's capacity. Using interfaces instead of multiple edges between ASes provides a simple graph, instead of an equivalent multigraph, which is closer to N-Tube's specification.

We assume an inter-domain *control plane* that enables each AS to obtain multiple loop-free paths to reach a destination AS (see, e.g., [59, Chapter 7]). These paths are expressed at the granularity of interfaces between the ASes. We also assume that clocks are loosely, globally synchronized, 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 is no constraint on the distribution of compromised ASes. We consider malicious ASes that can collude and attempt to allocate excessive amounts of bandwidth in order to exhaust the available bandwidth to be shared with other ASes, e.g., by requesting 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., by spoofing a signed message without the appropriate private key. Hence, attackers can replay

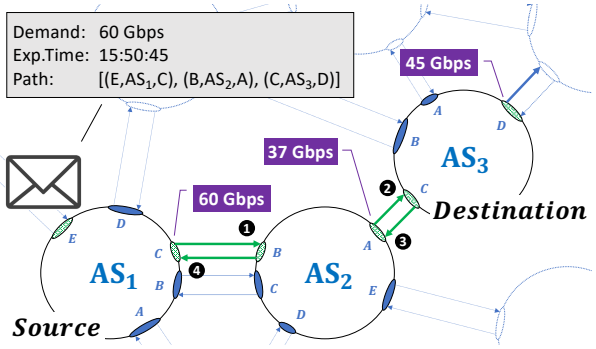


Figure 1: The process of making a reservation

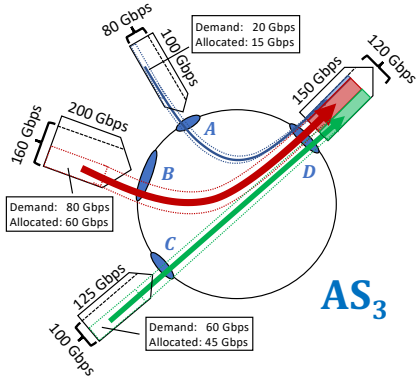


Figure 2: Bandwidth allocation computation at AS_3 distributes the egress link's (adjusted) bandwidth capacity D proportionally to three ingress demands.

legitimate reservation requests, but cannot craft new ones for ASes they do not control.

Our modeling is at the granularity of ASes and excludes end hosts within ASes. We consider the case of malicious end hosts within off-path ASes to be subsumed by the stronger case where their entire ASes are malicious. We assume that all on-path ASes and their end hosts are honest. Honest ASes are expected to refrain from allocating excessive bandwidth due to the associated costs and to filter such requests from malicious hosts inside them using separate mechanisms.

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 paper's scope. Such attackers could execute DoS attacks by ignoring reservations and blocking data traffic directly. We also do not consider data-plane attacks where an adversary sends excessive traffic or excessive amounts of bandwidth requests. If a malicious AS sends more traffic than what the algorithm has allocated, an enforcement mechanism detects and blocks that traffic from the misbehaving AS. Moreover, flooding the bandwidth reservation algorithm with reservation requests can be easily prevented by limiting the frequency of requests per AS.

3 N-TUBE OVERVIEW

Our N-Tube algorithm enables ASes to reserve bandwidth on network paths by reserving bandwidth on each inter-domain link on the path. A reservation consists of a path, an expiration time, and a bandwidth amount. 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 to adjust their reservation paths and demands.

To reserve bandwidth, the source AS chooses loop-free paths to the destination AS (obtained from the control plane, see Section 2.2), an amount of bandwidth and an expiration time, combines them in a reservation message, and authenticates it, e.g., with RPKI [48]. Figure 1 illustrates the reservation process. The source AS_1 sends a reservation message demanding `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 of the accumulated bandwidths, i.e., `37 Gbps`.

N-Tube has two user-specific parameters. First, N-Tube enforces an upper bound, denoted by $maxT \in \mathbb{N}$, on how long the expiration time can be set into the future. This forces ASes to update their reservations regularly, roughly every 5 minutes. Second, N-Tube only reserves a fixed portion δ ($0 < \delta < 1$) of each link's total capacity, called the *adjusted capacity*. 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.

3.1 Bounded Tube Fairness (G5)

The main challenge for a resource allocation algorithm is to treat all reservations *fairly*, and to provide a *lower-bounded* bandwidth allocation for honest ASes, even when adversaries try to congest a link by demanding excessive bandwidth.

To provide *fair* bandwidth allocation, N-Tube bounds excessive demands by the links' capacities, and shares the resulting demands proportionally. This is illustrated in Figure 2 by the bandwidth allocation computation at AS_3 in the above example:

(1) AS_3 factors the demands converging at a given egress interface by each ingress interface. These factored demands are called *tubes*, as we visualize them this way.
(2) AS_3 bounds the accumulated demand of each tube by its ingress and egress links' adjusted capacities, which we call their *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 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 $\delta = 0.8$ in this case, and are indicated by the dotted lines. The three demands of `20 Gbps`, `80 Gbps`, and `60 Gbps` 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, respectively.

3.2 Minimum Bandwidth Guarantee (G4)

By bounding the accumulated demands of tubes in the second step of the bandwidth allocation computation, we guarantee that each tube obtains 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 demands inside it.

We illustrate how N-Tube computes bandwidth allocations in the presence of adversaries with three examples. We assume that all ASes on the given path are honest, and any AS off this path may be adversarial. The goal of the adversaries is to reduce as much as possible the allocated bandwidth for the honest ASes between interface C and interface D. Hence, we allow adversaries 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 Figure 3 example, two adversarial ASes demand in total `400 Gbps` (`150 Gbps` and `250 Gbps`, respectively) of bandwidth through 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 B.

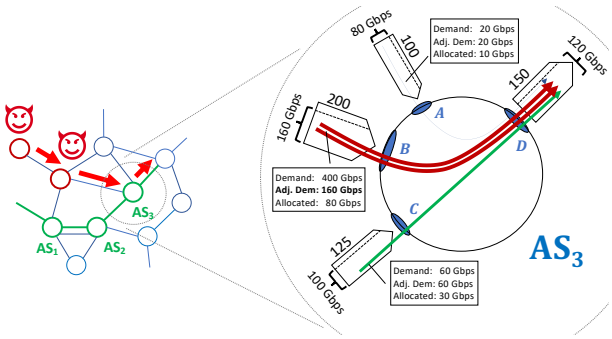


Figure 3: Limit demands on the ingress interface.

Limit each AS's demands by the egress link's capacity: In Figure 4, an adversarial AS demands in total `200 Gbps` to interface D: `80 Gbps` and `120 Gbps` through interfaces A and B, respectively. However, since its combined demand of `200 Gbps` exceeds the egress link's capacity, N-Tube reduces both demands proportionally by a *scaling factor*. The scaling factor is the ratio of the egress link's adjusted capacity to the total adjusted demand of the adversarial AS, i.e., `120/200` = 0.6. This results in the *reduced demands* of `48 Gbps` (= $0.6 \cdot$ `80 Gbps`) and `72 Gbps` (= $0.6 \cdot$ `120 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. 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 interface C, and the reduced demands of `48 Gbps` and `72 Gbps`, from interfaces A and B. The computed allocations are therefore `40 Gbps`, `32 Gbps`, and `48 Gbps`, respectively.

Worst case, minimum bandwidth guarantees: In the Figure 5 example, all off-path ASes are adversarial and demand as much as

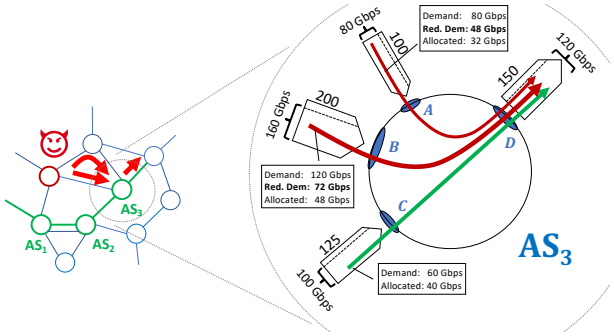


Figure 4: Limit demands on the egress interface.

they can on all the ingress links, i.e., a maximum of `80 Gbps` on interface A and `160 Gbps` (`40 Gbps` and `120 Gbps`, respectively) on interface B. This represents a worst-case attack: even when more adversarial ASes are present, their bandwidth demands will be adjusted, and 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 benign demand of `60 Gbps` from interface C. Hence, this benign demand cannot be reduced to less than `24 Gbps` by any amount of external demands. This provides the minimum bandwidth guarantee for the honest reservation at AS_3.

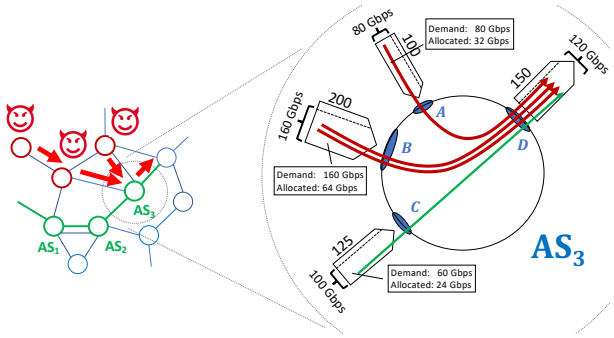


Figure 5: Worst-case minimum bandwidth guarantee.

Global lower bound: By applying the idea from the previous example, we can provide a *local lower bound* llb_3 for the proportion of D's adjusted link capacity that can be allocated to the benign demand. In the worst case, all tubes have excessive demands at interface D. Then llb_3 is given as the ratio of the benign demand of `60 Gbps` and the accumulated adjusted capacities of all ingress links, i.e., $llb_3 = 0.17 \approx$ `60 Gbps/(100 Gbps+160 Gbps+80 Gbps)`. Likewise, we can compute llb_1 and llb_2 with respect to AS_1 and AS_2's ingress links' capacities. Note that these local lower bounds do not depends on the adversarial demands.

The *request ratio* $reqRatio_0$ at the source AS_1 is defined as the ratio of the benign demand of `60 Gbps` and the total demand of reservations starting at interface E of AS_1 (see Figure 1). The global lower bound glb for the bandwidth that can be allocated to the

honest demand is derived as

$$glb = reqRatio_0 \cdot llb_1 \cdot llb_2 \cdot llb_3 \cdot 120\ \texttt{Gbps}.$$

Note that *glb* only depends on the request ratio at the (honest) source and the capacities of the on-path ASes' ingress links. The request ratio is bounded by the number of reservations the source AS_1 starts at its interface E, i.e., under its own control, and is not influenced by (malicious) reservations (see Appendix B for details). This is, intuitively, why N-Tube's bandwidth allocation computation provides (G4). Furthermore, the computation splits the adjusted link's capacity proportionally between the non-excessive demands. This illustrates, informally, that N-Tube provides (G5) for each link.

3.3 Stability (G3)

N-Tube's upper bound *maxT* on the expiration time forces ASes to renew their reservations regularly. This allows N-Tube to stabilize the allocations in a predictable time period *stabT* of constant demands after a burst in demands. The time period *stabT* needed to stabilize demands can be shown to be the product of *maxT* and the length of the longest reserved path $\hat{p}$ in the network, i.e.,

$$stabT = length(\hat{p}) \cdot maxT.$$

Intuitively, the bandwidth computation at the first AS only depends on requested demand at that AS and the constant bandwidth allocations are then successively propagated to all ASes on the path at each renewal of the reservation. This provides an informal argument that N-Tube achieves the *stability* property (G3).

We will show that, after the entire network stabilizes, N-Tube's bandwidth allocations also satisfy *bounded tube fairness* (G5).

3.4 Immutability (G2) and Availability (G1)

By reserving only a fraction δ of the available bandwidth, N-Tube can always provide a positive (but possibly small) amount of bandwidth for a new reservation. This ensures *availability* (G1). Unused bandwidth capacities can be used for best-effort traffic. N-Tube does not change established reservations until they either expire or are explicitly deleted by the source (see Section 4). This yields *immutability* (G2).

4 ALGORITHM DETAILS

In this section we first define the network model, messages, reservation maps, and N-Tube's dynamics. We then specify its bandwidth allocation computation, and describe the local properties it achieves. For notation see Appendix C and for full details Appendix D.

4.1 Network and Messages

Network. We model the *network* as a weighted, directed graph (N, E, cap), for which we give a simplified definition:

- The nodes N are given by the set $V \times I$, where V is a finite set of vertices (ASes), 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, respectively. 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 $cap(v, i)$).

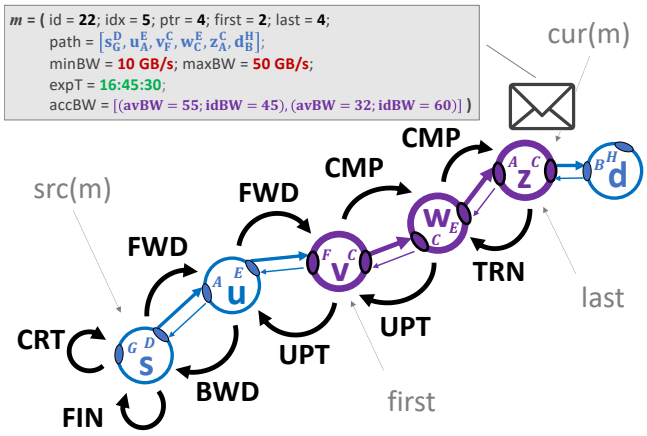


Figure 6: The message m is processed along p by the events described in Section 4.4.2.

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

$$\mathcal{P} = \left[\langle\!| \, inI \in I; as \in V; egI \in I \, |\!\rangle\right],$$

where $[A]$ is the type of lists with elements in A and $\langle\!| \ldots |\!\rangle$ is our notation for records. 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) it is *loop-free*, i.e., each AS occurs at most once on the path.

In what follows, we denote the elements of sets V and I with lowercase letters: for V we use the letters s, x, v, and z, for ingress interfaces i and i', and for egress interfaces e and e'. We also write the ingress interface as subscripts and the egress interface as superscripts to the AS identifier, e.g., x_i^e.

Messages. There are two types of messages: *reservation* and *deletion messages*. The message fields 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 6:

- 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) indicating which version of the reservation the messages refers to. Version indices are used to update existing reservations (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) refer to the first and last AS on p, respectively.
- The minimum $minBW \in \mathbb{R}_0^+$ (= 10 GB/sec) and maximum bandwidth $maxBW \in \mathbb{R}_0^+$ (= 50 GB/sec) state the range of bandwidth the source AS would like to reserve. Hereby, $maxBW$ states the source's *demand* in the 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 4.3.

The functions *src*, *first*, *cur*, and *sgmt* on valid messages 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* (including the endpoints), respectively. In the example of Figure 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.

The type of *messages* $\mathcal{M} = \mathcal{M}_R + \mathcal{M}_D$ is the disjoint sum of *reservation messages* $\mathcal{M}_R$ and *deletion messages* $\mathcal{M}_D$. A message m is *valid*, if $m.path$ is a valid path, and for its pointers it holds that $0 \leq ptr, first, last \leq length(m.path)$ and $first < last$. The bandwidth range must be a non-empty interval, i.e., $0 \leq m.minBW \leq m.maxBW$ and $m.maxBW > 0$, and thus only non-zero bandwidth allocations are allowed.

4.2 Message Processing

Reservation Maps. Each AS maintains its own reservation map where all currently valid reservations with a path traversing this AS are stored. 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*, 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 6 is

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

N-Tube allows source ASes to flexibly update their reservations multiple times before they expire, and therefore stores different versions of each reservation in the *version map 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 on p, the expiration time $expT$ given by m, and the reserved bandwidth $resBW$ determined by N-Tube. We call the reservation's entries in the version map its *versions*, e.g., for m

$$verM(5) = (\!| minBW = 10 \text{ GB/s}; maxBW = 50 \text{ GB/s}; \\ idBW = 60 \text{ GB/s}; resBW = 32 \text{ GB/s}; \\ expT = 16{:}45{:}30 |\!).$$

A version vr is *currently valid* at time t, written $cvalid(vr, t)$, if it is *successful*, i.e., $vr.minBW \leq vr.resBW$ and *not expired*, i.e., $vr.expT \geq t$. 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 as the source can send traffic using any existing version of its reservations. In this way, sufficient bandwidth is guaranteed to be available in the worst case.

Reservation Process. N-Tube processes a reservation message depending on its direction and position on the path. As shown in Figure 6, suppose that AS s intends to make a new reservation *id* on a path p. It then 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 with 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 (see Section 4.3 for the details of the computation),
(3) updating m to a new message m' by appending the computed results to $accBW$ and by 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 (indicated by pointer *last*) on the path has processed m, it sends the message m' backward (TRN). During the backward traversal, each AS on the path extracts how much bandwidth $finBW$ could be reserved on the entire path by taking the minimum of $maxBW$ and $accBW$, i.e., what have been computed in the forward traversal

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

Analogously to the forward traversal updates, an AS updates (UPT) its reservation map according to the same two cases: (i) the reservation was successful, i.e., $m.minBW \leq finBW$, 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., $m.minBW > finBW$, and each AS deletes the corresponding version from its reservation map. The ASes between *first* and source AS s simply send the message backwards (BWD) without processing it until s finally receives it (FIN).

Renewal and Deletion. If s intends to *renew* one of its reservations, it sends a new reservation message m, containing an updated bandwidth range and expiration time, along the previous path p. To

delete a reservation's version, a source AS sends a *deletion* message along the corresponding path.

4.3 Fair Bandwidth Allocation

The heart of the N-Tube algorithm is its bandwidth allocation computation. We assume that a *valid* reservation message m was sent by its source AS s and arrives at an AS x lying between *first* and *last* on m's path at the current time t. First, N-Tube derives its source s (= $src(m)$) and its current AS, ingress, and egress interfaces x_i^e (= $cur(m)$). Given m and $resM_x$, the bandwidth allocation 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 is allocated to s with respect to all active reservations in $resM_x$ between interfaces i and e.

The corresponding functions *avail* and *ideal* are defined below. To simplify notation, we fix the message m and its elements s, id, x, i, and e, and omit $resM_x$, t, and the parameter δ as arguments. The functions *resSr*, *resEg*, and *resIn* extract a reservation's source AS and the current AS' egress and ingress interfaces, respectively. For full details, see Appendix D.5.

4.3.1 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

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

It subtracts the aggregated *allocated bandwidth* of all currently valid reservations with egress interface e from the link's total capacity $cap(x,e)$, and multiplies the result with the parameter $0 < \delta < 1$. The factor δ guarantees available bandwidth for subsequent reservations.

4.3.2 Limiting Excessive Demands. To avoid that s reserves more bandwidth in one request than physically available, N-Tube limits the bandwidth demand $demBW(r)$ of a reservation r by the ingress and egress links' adjusted capacities. The resulting *requested demand* of a reservation r is defined by

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

As illustrated in Section 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 of 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 e

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

We analogously define the *ingress demand* on interface i.

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

$$adjReqDem(r) = \min\{inScalFctr(s,i), egScalFctr(s,e)\} \cdot reqDem(r,i,e)$$

with s, i, and e the corresponding source AS, ingress and egress interface of r, respectively. We compute for source AS s the *egress scaling factor* on the egress link e as the source's proportion of the total *egress demand* bounded by the egress link's capacity, given by

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

We analogously define the source's *ingress scaling factor*.

4.3.3 Ideal Bandwidth Computation. Given a message m with x_e^i on its path, the function *ideal* computes how the adjusted capacity $\delta \cdot cap(x,e)$ of the egress link e is shared in a so-called *bounded tube fair* manner among all existing reservations (in $resM_x$) with the same egress link e:

$$ideal(s,id,i,e) = reqRatio(s,id,i,e) \cdot tubeRatio(i,e) \cdot \delta \cdot cap(x,e).$$

This (1) proportionally splits the egress link's adjusted capacity between all ingress links by multiplying it with *tubeRatio*, and (2) further splits the result proportionally between all reservations from i to e by multiplying it with *reqRatio*.

We define these two ratios in the following.

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 $\min\{cap(x,i), tubeDem(i,e)\}$, and the aggregated bounded tube demands at e

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

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

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

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

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

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

The function *adjIdDem* is defined similarly to *adjReqDem* but for the previously computed ideal bandwidth, and *transitDem* is the aggregation of all *adjIdDem*'s at ingress interface i.

4.4 Formalizing N-Tube

We formalizes N-Tube using labeled transition systems. A *labeled transition system* (LTS) $\mathfrak{T} = (\Sigma, \Sigma_0, \Lambda, \Delta)$ consists of a state space Σ, a set of initial states $\Sigma_0 \subseteq \Sigma$, a set of labels Λ, also called *events*, and a (labeled) transition relation $\Delta \in \Lambda \to \mathbb{P}(\Sigma \times \Sigma)$. *Executions* of $\mathfrak{T}$ 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 in an initial state, i.e., $\sigma_0 \in \Sigma_0$, and progresses according to the transition relation Δ, i.e., for all $n \in \mathbb{N}$, $(\sigma_n, \sigma_{n+1}) \in \Delta(\lambda_n)$.

Below, we fix the environment: a network graph (N, E, cap) as in Section 4, a partition $V = H \uplus M$ (with H and M denoting sets of honest and malicious ASes), and the fraction $0 < \delta < 1$ of the links' adjusted capacities. We model the behavior of both honest and malicious ASes. See Appendix D for full details.

4.4.1 States. We define the set of *states* Σ as the record

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

A *state* $\sigma \in \Sigma$ describes a snapshot of the system at a given point in time, given by its *time* field. In our model, we assume discrete time, which is *loosely* synchronized between all ASes, 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 field *buf* of type $Buff = V \times I \to \mathbb{P}_{fin}(\mathcal{M})$ models network *buffers*, where $buf(x, i)$ holds the set of messages arrived at interface $i \in I$ of AS $x \in V$. The field *res* models all ASes' *reservation maps* as presented in Section 4.2. Finally, the field *kwl* models the *attackers' knowledge*: the set of messages created, collected, and shared by malicious ASes.

4.4.2 Events. The set of events Λ consists of system events and environment events. System events formalize the N-Tube algorithm: its *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 6). This results in seven events describing reservation message processing, three handling deletion messages, and one event for dropping messages. Three events model the system's environment: a *time progress* (clock tick) event and two events modeling the *attackers' capabilities*. Below, we present the attacker events and one representative message processing event.

Attacker Events. Malicious ASes can execute two events: (i) receive a message, partially modify it, and store the resulting message in the attackers' knowledge *kwl*, and (ii) send a message in *kwl* to any neighbor AS in the network. Recall that *kwl* also includes any message in $\mathcal{M}$ with a malicious source AS.

We discuss the former event, *CLT*, in more detail. It is defined by

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

Here, an attacker $a \in M$ receives a message m from his buffer $buf(a, i)$ at interface i, possibly modifies it, and adds the resulting message to *kwl*. The equivalence relation $m \approx m'$ expresses that m and m' coincide except on their mutable fields *ptr* and *accBW*. This models our assumption that, in an N-Tube implementation, the source AS signs the immutable fields with its private key, while the mutable fields remain unprotected.

The two events model powerful attack capabilities. Malicious ASes can anytime make arbitrary reservation requests from their own ASes, partially modify observed requests, replay old messages, and collude through out-of-band channels to share their knowledge and synchronize attacks. However, attackers cannot spoof messages from honest ASes, modify reservations stored in the reservation maps of honest ASes, or change the system's global time.

The CMP Event. For each (parametrized) event, we define a relation on states σ and σ', which is described by *guard* predicates (describing the event's executability) and state update *actions*. Here is the definition of the CMP event:

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

Upon receiving a reservation message m at interface i (first guard) at time t (second guard), AS v allocates bandwidth using the function *save* (first action), and forwards the modified reservation message m' using the function *forward* (second action). All unmentioned fields remain unchanged. Guards (3–5) ensure that m is well-formed and compatible with existing reservations in v's reservation map that corresponds to m. Guard (6) determines whether v is on the path segment, i.e., m's pointer is between *first* and *last*. Guard (7) 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 last guard models the computation of the modified message m', using the function *compute* to update of received message m's *accBW* field.

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

This function determines the available and ideal bandwidths that AS v can allocate using the functions *avail* and *ideal* from Section 4. The results are appended to m's *accBW* field.

5 PROPERTIES

In this section we first define the notions of valid executions, successful reservations, and constant demands that we use to specify N-Tube's global properties (G1–G5). We refer to Appendix E for details.

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

Table 1: Formalization of properties (G1–G5)

Property	Formula
(G1) Availability: If an honest AS s makes a successful reservation m at time t, then some non-zero bandwidth will be reserved on its path until it expires.	$\forall m \in \mathcal{M}_R, s \in V, t \in \mathbb{N}, n \in \mathbb{N}, v \in sgmt(m).$ $Succ(s, m, t) \wedge \sigma_n.time \in]t; m.expT]$ $\Rightarrow \sigma_n.res_v(s, m.id).vrs(m.idx).resBW > 0$
(G2) Immutability: If an honest AS s makes a successful reservation m at time t, the reserved bandwidth stays the same for all ASes on its path until it expires.	$\forall m \in \mathcal{M}_R, s \in V, t \in \mathbb{N}, n, n' \in \mathbb{N}, v, v' \in sgmt(m).$ $Succ(s, m, t) \wedge \sigma_n.time, \sigma_{n'}.time \in]t; m.expT]$ $\Rightarrow \sigma_n.res_v(s, m.id).vrs(m.idx).resBW = \sigma_{n'}.res_{v'}(s, m.id).vrs(m.idx).resBW$
(G3) Stability: If constant demands D between t_0 and t_1, then all reservations allocate the same amount of bandwidth from $t_0 + stabT$ until t_1.	$\forall D \in \mathcal{D}, t_0, t_1 \in \mathbb{N}, n, n' \in \mathbb{N}, r, r' \in Res, v \in H, m \in rng(D).$ $Stab(D, t_0, t_1) \wedge \sigma_n.time, \sigma_{n'}.time \in]t_0 + stabT; t_1] \wedge r = \sigma_n.res_v(src(m), m.id) \wedge r' = \sigma_{n'}.res_v(src(m), m.id)$ $\Rightarrow allocBW(r, \sigma_n.time) = allocBW(r', \sigma_{n'}.time)$
(G4) Minimum Bandwidth Guarantee: For constant demands D between t_0 and t_1 and for any honest AS's reservation, there is a lower bound on the allocated bandwidth that only depends on the request ratio on the first link, a factor G depending on the path's link capacities, and $m.maxBW$.	$\forall D \in \mathcal{D}, t_0, t_1 \in \mathbb{N}.\ Stab(D, t_0, t_1)$ $\Rightarrow \exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + stabT$ $\wedge\ \forall m \in rng(D), r \in Res.\ \exists G > 0.\ \forall n > \tilde{n}, v \in sgmt(m).$ $\sigma_n.time \in]t_0 + stabT; t_1] \wedge r = \sigma_n.res_v(src(m), m.id)$ $\Rightarrow allocBW(r, \sigma_n.time) \geq G \cdot reqRatio(m, \sigma_{\tilde{n}}.res_{first(m)}) \cdot m.maxBW$
(G5) Bounded Tube Fairness: For constant demands D between t_0 and t_1, in the absence of congestion, bandwidth of egress links is allocated proportionally between tube demands and, in case where tube demands exceed their ingress links' capacities, their tube ratio is bounded.	$\forall D \in \mathcal{D}, t_0, t_1 \in \mathbb{N}.\ Stab(D, t_0, t_1)$ $\Rightarrow \exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + stabT$ $\wedge\ \forall m \in rng(D), v \in sgmt(m), i, i', e \in I, n > \tilde{n}.$ $tubeDem_v(i, e) \in]0; \delta \cdot cap(v, i)] \wedge tubeDem_v(i', e) \in]0; \delta \cdot cap(v, i')]$ $\Rightarrow \frac{tubeRatio_v(i, e)}{tubeRatio_v(i', e)} = \frac{tubeDem_v(i, e)}{tubeDem_v(i', e)}$

Time-Progress: Global time always eventually advances

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

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

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

This is satisfied if (i) all honest ASes run a fair scheduling algorithm (e.g., Round-Robin) to prevent message starvation, and (ii) messages are dropped in case of buffer overflow.

Properties (G1) and (G2) 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, i.e., there exist $n \in \mathbb{N}$ and $i \in I$ such that
$$\lambda_n = FIN(m, s, i, t) \wedge finBW(m) \geq m.minBW.$$
- *No deletion:* There is no deletion event matching the reservation $(src(m), m.id)$ and version $m.idx$ before m's expiration time.

We abbreviate this definition by the predicate $Succ(s, m, t)$.

For properties (G3–G5) we model "constant bandwidth demands" as a partial function $D \in \mathcal{D}$ with

$$\mathcal{D} = V \times \mathbb{N} \rightharpoonup_{fin} \mathcal{M}_R$$

such that $D(s, id) = m$ implies $src(m) = s$ and $m.id = id$. We say that a reservation message m *corresponds to* D if $(src(m), m.id) \in supp(D)$ and m coincides with $D(src(m), m.id)$ on all fields except ptr, $expT$, $minBW$, and $accBW$. The *stabilization time*

$$stabT = \max\{length(m.path) \mid m \in rng(D)\} \cdot maxT$$

is the maximal time that the reservation requests in $rng(D)$ must be renewed along their paths to reach a stable state.

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

- $t_1 \geq t_0 + stabT$,
- for all $(s, id) \in supp(D)$, the source AS s has successfully made a reservation confirmed by a message m corresponding to $D(s, id)$ 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 D, and
- there are no deletion events between t_0 and t_1 for reservations given by $supp(D)$.

We abbreviate this definition by the predicate $Stab(D, t_0, t_1)$.

Given a valid execution, we formally specify the global properties (G1–G5) in Table 1 and state our main theorem.

Theorem (N-Tube's qualitative properties)**.** Our LTS model of N-Tube satisfies properties (G1–G5).

The inductive proofs that our LTS model of N-Tube satisfies these properties are in Appendix E.

6 STATISTICAL ANALYSIS OF N-TUBE

Our qualitative analysis of N-Tube by inductive proof in Section 5 establishes the desired correctness and security guarantees, but offers no insight into the actual dynamics of these guarantees. We therefore additionally conduct quantitative measurements about these guarantees. In particular, we use Maude-based simulation [23] and statistical model checking (SMC) [62] to analyze N-Tube with respect to properties (G1–G5).

Our goal is twofold: (i) to validate our inductive proofs via independent machine-checked statistical verification; and (ii) to explore quantitative aspects of N-Tube in various adversarial scenarios, w.r.t. stability, fairness, and resistance to malicious power, via statistical estimations, which goes beyond the inductive proofs.

6.1 Why Maude and SMC?

Quantitative system analysis typically requires an executable artifact. As rewriting logic [55] is a generic framework for specifying the semantics of a wide range of computation models, LTSs can be naturally expressed as *rewrite theories* in it, and executed as *system modules* in Maude [23]. A rewrite theory consists of an equational theory, specifying the system's data types, and a collection of *labeled conditional rewrite rules* of the form `crl [l] : t => t' if cond`, where l is a label. Such a rule specifies a transition from a system state, represented by the term t, to a new state t', provided the condition *cond* holds.

The Maude ecosystem supports highly performant, machine-checkable, automated formal analysis, including simulation and SMC [23, 62]. In particular, compared to conventional simulations or emulations, SMC can verify a property specified, e.g., in a *stochastic temporal logic*, up to a *statistical confidence level* by running Monte-Carlo simulations of the system model. The expected value $\bar{v}$ of a property query belongs to the interval $[\bar{v} - \frac{\beta}{2}, \bar{v} + \frac{\beta}{2}]$ with $(1 - \alpha)$ statistical confidence, where parameters α and β determine when an SMC analysis stops performing simulations [62].

The Maude system has been very successful in analyzing high-level designs of a wide range of distributed and networked systems [16, 50, 51, 68, 72, 73]. In particular, Maude-based validation using SMC provides additional confidence about claimed statements by analyzing large parameter spaces. Moreover, with significantly less effort than the actual system implementations (e.g., 20x less in terms of lines of code [52]), SMC-based performance estimations and predictions (in an early design stage) have shown good correspondence with implementation-based evaluations under realistic deployment [16, 52].

6.2 Model Transformation

We can directly express N-Tube's LTS from Section 4.4 as an untimed, nondeterministic, and non-probabilistic rewrite theory. This is, however, not well-suited for quantitative analysis by SMC. Therefore, we (manually) transform this rewrite theory into a *timed, purely probabilistic* rewrite theory by following the systematic methodology in [3]. In particular, the transformation assigns to each message a delay sampled from a *continuous probability distribution*, which determines the firing of the rewrite rule receiving the message. The resulting model is free from unquantified nondeterminism and can be simulated by the original model. Both the original and the transformed probabilistic Maude specifications of N-Tube are available at [1].

The system state of the transformed model consists of a *multiset* of *objects* and *messages*. An object of class C is represented as a term $< o : C \mid att_1 : val_1, \ldots, att_n : val_n >$, with o the object's identifier, and val_1 to val_n the current values of attributes att_1 to att_n. An incoming message of the form `{t, msg}` is ready to be consumed at the global time t, while an outgoing message of the form `[t + d, msg]` will be delivered in d time units after t where the message delay d is sampled from some continuous probability distribution (e.g., the lognormal distribution). Each message *msg* has the form `to o from o' : mp`, with o, o', and *mp* the message receiver, sender, and payload, respectively. A *scheduler* object is also specified to advance the global time, and to deliver outgoing messages at the specified times.

N-Tube's dynamic behavior is axiomatized by specifying its transition patterns as rewrite rules. Consider the message processing event *CMP* in Section 4.4.2. The following transformed conditional rule `[cmp]` specifies that, upon receiving a reservation message `res(M)` at global time `T`, an AS `O` updates its local reservation map accordingly (by the `save` function), and forwards the modified message (by the `compute` function) to `next` hop in the path:

```
crl [cmp] :
   {T,  to O from O' : res(M)}
   < G : Table | links : LS, ATTS' >
   < O : As | resMap : RM, ATTS >
=>
   < G : Table | links : LS, ATTS' >
   < O : As | resMap : save(M,O,RM,LS,AVL,IDL), ATTS >
   [T + md, to next(O,M) from O : compute(M,AVL,IDL)]
   if (atSrt(M) or onPth(M)) /\ pathCheck(M)
      /\ resMsgCheck(M,T) /\ resMapCheck(M,RM)
      /\ AVL := avail(LS,O,RM,T,M)
      /\ IDL := ideal(LS,O,RM,T,M) .
```

where the network topology and all `links`' capacities are stored in a global "table" `G`. The message delay `md` is probabilistically sampled from a predefined distribution. The functions `avail`, `ideal`, `save`, and `compute`, as well as the predicates in the condition, are defined following Section 4.4. The variables `ATTS` and `ATTS'` refer to the rest of attributes that do not affect the next state.

6.3 Statistical Analysis

We investigate the following questions about N-Tube using our statistical analysis:

- Are the statistical verification results consistent with our hand-written inductive proofs of (G1–G5)?
- How does N-Tube actually perform in worst- and average-case malicious scenarios w.r.t. stability and fairness? In particular, how does it resist increasing attack power?

6.3.1 Benchmark. To statistically analyze N-Tube's properties we implement three *parametric* generators: a topology generator (TG), a path generator (PG), and a workload generator (WG). We use these to probabilistically generate a different initial state for each simulation in an SMC analysis. Specifically, TG generates *scale-free* Internet topologies with strongly connected ASes, which is also characteristic of CAIDA's realistic AS-level Internet graph [19].

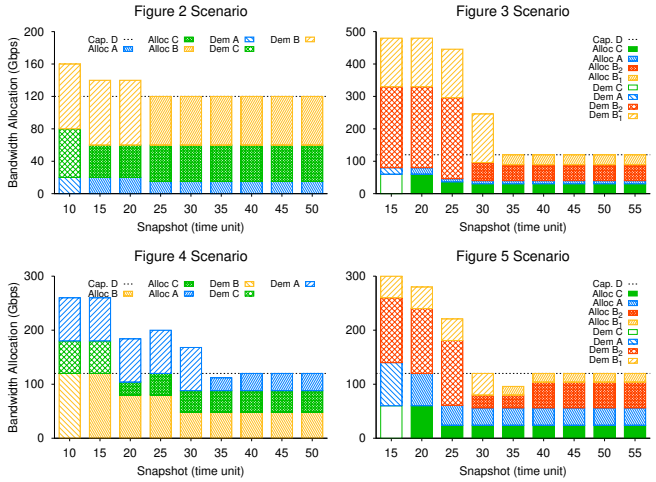


Figure 7: Measuring stability and fairness. Time units are defined as logical clock ticks in our probabilistic model.

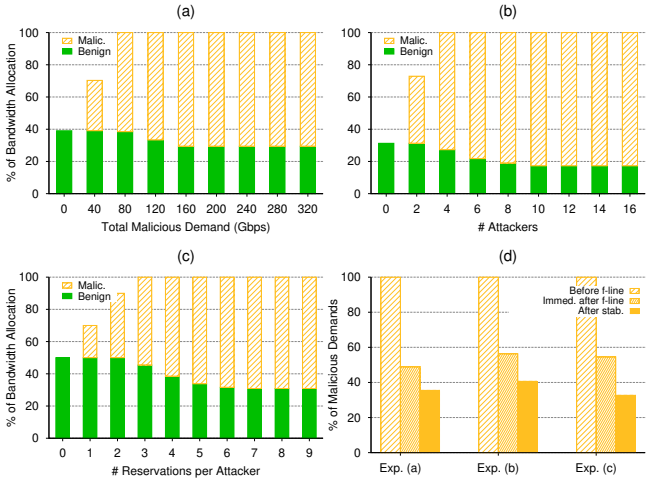


Figure 8: Measuring the impact of increasing malicious power.

Each link between nodes is assigned a bandwidth probabilistically sampled from an interval. PG then explores the generated graph, and collects paths from *sources* to *destinations*. WG provides the generated sources, including adversaries, with reservations, renewals, and deletions on a probabilistic basis, where each of these three types of requests is parametric in the algorithm-specific parameters (such as *maxBW*, *expT*, etc.). See Appendix F for a complete list of the generators' 18 parameters and their default values.

6.3.2 Experimental Setup. To mimic the real-world network environment, we use the lognormal distribution for probabilistically sampling message delays [14]. We employed a cluster of 50 d430 Emulab machines [75], each with two 2.4 GHz 64-bit 8-Core E5-2630 processors, to parallelize SMC with the PVeStA tool [4] (part of the Maude ecosystem). We set the statistical confidence level to 95% and the size parameter to 0.01 for all our experiments.

6.3.3 Analysis Results. We have subjected the transformed Maude model to the above generators and PVeStA, and performed three sets of experiments according to our experimental goal with 100 ASes by default. Each simulation or SMC analysis took up to three hours (worst case) to terminate.

Exp. 1: Verifying Properties. In all our SMC analyses, the probabilities of satisfying N-Tube's global properties (G1–G5) are 100%. This provides a strong independent *validation of our proofs* for the properties, and of the *model transformation* (from the LTS model into Maude), via machine-checked analysis.

Exp. 2: Stability & Fairness. We report the simulation results for the scenarios in Section 3. Figure 7 depicts the bandwidth reservations at interface D and their state, demanded (striped bars) or allocated (solid bars), as a function of (simulation) times where we take "snapshots" of the system state. The demanded bandwidth is replaced by an actual allocation once the reservation succeeds. These allocations adapt over time to self-renewals and other demands. For the scenarios in Figures 3 and 5, we individually measure the allocated bandwidth for each of the two demands (B_1 and B_2) through interface B. In all scenarios, the allocations in the entire network *converge* and *stabilize*; the total allocations are always *bounded* by D's adjusted capacity (dashed line), and distributed *proportionally* to the demands after stabilization as expected (Section 3). Hence, from the quantitative perspective, these results further demonstrate stability and fairness, in particular for the *worst-case scenarios* (Figures 3–5) where attacks are mounted directly on an honest path.

Exp. 3: Impact of Increasing Malicious Power. To analyze the influence on bandwidth allocation of increasing malicious power, we randomly positioned the attackers in the network (not necessarily neighbors of the targeted path), and picked a relatively small number of destinations (5 out of 100 ASes) for the reservations so that all demands, including malicious ones, *converge to these destinations*. We then randomly selected one destination and one of its egress interfaces, and reported the associated aggregate allocations for the benign sources and attackers, respectively.

Figure 8 (a)–(c) show the allocation percentage as a function of attacker capability, represented by *total demanded bandwidth*, *number of attackers*, and *number of issued reservations per attacker*, respectively. With increasing malicious power, the attackers tend to occupy more bandwidth until the entire allocation *stabilizes* (starting from 160 Gbps, 10 attackers, and 7 reservations/attacker, respectively); thereafter their demands are adjusted, and thus limited by the *links' capacities* and *scaling factors*. These results further provide quantitative assessments of N-Tube with varying malicious powers by exploring the large parameter space.

We also measured the adversaries' allocated bandwidth reduction by N-Tube's "frontline defense". A *frontline defender* is the first honest AS on an attacker reservation's path that can mitigate the impact of the attack by limiting the adversary demands. We divided the timeline into three phases: (i) before malicious demands reach the frontline defender; (ii) immediately after those demands break through the frontline; and (iii) after the network stabilizes.

Figure 8 (d) reports for Exp.(a)–(c) the percentage of original malicious demands that was allocated to the adversaries in each phase. Phase (iii)'s computation is based on the minimum stabilized "point" (e.g., 160 Gbps in Exp.(a)): The higher the stabilized point

is that we consider, the more reduction there will be. As demonstrated in Exp.(d), N-Tube's frontline defense plays an important role in limiting the adversarial demands, e.g., in Exp.(a) ∼50% of the malicious demands can be reduced, which constitute almost 80% of the total reduction.

7 RELATED WORK

7.1 Quality of Service and DDoS Protection

Congestion Control enables end-to-end connections possibly with multiple paths, to control their path rates to fairly mitigate congestion. However, this approach is based on per-flow fairness with complete knowledge of users' utility functions. In contrast, we take the stance that new Internet architectures [34, 59, 77] 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 misbehavior (over time and per AS) and cannot revoke access.

Capability-Based Mechanisms [37, 46, 47, 53, 58, 76, 78] 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 legitimately acquire these tokens from numerous honest and malicious destinations, and send a huge number of low-volume flows through critical links.

Game-Based Mechanisms consider resource allocation for maximizing a global objective function as an "inverse game theory" problem: how to design games that achieve maximum social-welfare utility [39, 40]. 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 on their utility functions. Vickrey-Clarke-Groves (VCG) type mechanisms [71] 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 [39], which misses realistic attack scenarios. Furthermore, the main objective of these mechanisms is to increase efficiency, and not to provide minimal guarantees.

Resource Reservation Systems such as RSVP [82] and PCE [28][70] enable users to reserve bandwidth along network paths by allowing reservation state at the routers. However, they neither handle malicious reservations nor are their claims formally supported.

For path-based networks, a scalable inter-domain bandwidth allocation architecture, called SIBRA [9], was previously proposed. SIBRA is based on a distributed *bandwidth reservation algorithm* that processes reservation requests issued by autonomous systems (ASes) and allocates bandwidth on the path's links accordingly. Together with an *enforcement mechanism* that monitors and polices these reservations, SIBRA is claimed to provide effective QoS guarantees in general and *minimum bandwidth guarantees* (or *botnet-size independence*) in particular. However, only a high-level design is provided without a concrete algorithm or formal arguments to support the stated claims.

7.2 Formal Verification of Networking Systems

As we are not aware of any works applying formal verification to a bandwidth reservation system, we discuss here research in the broader area of network systems verification.

Qualitative Properties. One line of research has applied formal methods to verifying network configurations [6, 10–12, 29, 31, 33, 43–45, 49, 60, 66, 74]. Verified properties include reachability and loop freedom for a fixed, concrete network topology. More recently, NetSMC [79] supports stateful network verification of policies expressed in an LTL subset.

Other work studies secure networking protocols, including packet forwarding protocols [80], inter-domain routing protocols [20, 21], and routing protocols for mobile ad-hoc networks [7, 8, 13, 22, 24, 57], and verifies their security properties such as source authentication, path validation, route validity, and loop freedom. All these works analyze *qualitative* properties using techniques such as model checking, theorem proving, or hand-written proofs.

Quantitative Properties. Another critical aspect is the verification of a system's *quantitative* properties such as performance, rapid convergence, or the quick recovery from attacks. We focus on the analysis of DoS protection mechanisms. Meadows' cost-based framework [54] enables the (non-probabilistic) extension of existing protocol models and tools with cost accounting and comparisons [35]. It has been extended to cover timing aspects (e.g., slow DoS) and amplification DoS attacks [64, 68]. Approaches based on probabilistic or statistical model checking have been applied to analyze SYN flooding attacks on TCP/IP [2], the adaptive selective verification protocol [5, 25], and amplification attacks on DNS [27].

Recent research has developed domain-specific languages and tools for probabilistic network analysis. McNetKAT scales to large networks with thousands of nodes, but is limited to stateless models [65], while Bayonet [32] and Netter [81] support stateful models, but have a limited scalability. Neither of these languages and tools are expressive or scalable enough to represent and verify our N-Tube model with its complex state and arithmetic operations.

8 CONCLUSION

We have presented the design of N-Tube, along with the novel notion of bounded tube fairness. We developed formal models and verified all its safety and security properties. Moreover, we have gained: (i) additional confidence about our hand-written proofs via independent machine-checked statistical model checking, and (ii) a quantitative assessment of N-Tube's resistance to attacks by statistically exploring the large parameter space and varying malicious scenarios.

N-Tube is the first provably correct inter-domain bandwidth reservation algorithm and a major step towards a provably secure QoS scheme that also provides DDoS defense. The obvious next step is to build an efficient N-Tube implementation, as well as large-scale deployment by, e.g., proceeding along the lines proposed in [59, Chapter 10]. Preliminary results from an N-Tube prototype implementation, realized as part of the Colibri QoS infrastructure [61], have demonstrated N-Tube's deployability and scalability.

REFERENCES

[1] [n.d.]. *N-Tube: Secure Bandwidth Reservation in Path-Aware Internet Architectures.* Technical Report. https://github.com/anonymous-yokai/n-tube.

[2] Gul Agha, Michael Greenwald, Carl A. Gunter, Sanjeev Khanna, Jose Meseguer, Koushik Sen, and Prasannaa Thati. 2005. Formal Modeling and Analysis of DoS Using Probabilistic Rewrite Theories. In *International Workshop on Foundations of Computer Security (FCS '05).*

[3] Gul A. Agha, José Meseguer, and Koushik Sen. 2006. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. *Electr. Notes Theor. Comput. Sci.* 153, 2 (2006).

[4] Musab AlTurki and José Meseguer. 2011. PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool. In *CALCO (LNCS, Vol. 6859).* Springer, 386–392.

[5] Musab AlTurki, José Meseguer, and Carl A. Gunter. 2009. Probabilistic Modeling and Analysis of DoS Protection for the ASV Protocol. *Electron. Notes Theor. Comput. Sci.* 234 (2009), 3–18. https://doi.org/10.1016/j.entcs.2009.02.069

[6] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: semantic foundations for networks. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014,* Suresh Jagannathan and Peter Sewell (Eds.). ACM, 113–126. https://doi.org/10.1145/2535838.2535862

[7] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. 2011. Deciding Security for Protocols with Recursive Tests. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6803),* Nikolaj Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer, 49–63. https://doi.org/10.1007/978-3-642-22438-6_6

[8] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. 2014. Modeling and verifying ad hoc routing protocols. *Inf. Comput.* 238 (2014), 30–67. https://doi.org/10.1016/j.ic.2014.07.004

[9] Cristina Basescu, Raphael M Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. 2015. SIBRA: Scalable internet bandwidth reservation architecture. *arXiv preprint arXiv:1510.02696* (2015).

[10] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017.* ACM, 155–168. https://doi.org/10.1145/3098822.3098834

[11] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2018. Control plane compression. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018,* Sergey Gorinsky and János Tapolcai (Eds.). ACM, 476–489. https://doi.org/10.1145/3230543.3230583

[12] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2020. Abstract interpretation of distributed network control planes. *Proc. ACM Program. Lang.* 4, POPL (2020), 42:1–42:27. https://doi.org/10.1145/3371110

[13] Davide Benetti, Massimo Merro, and Luca Viganò. 2010. Model Checking Ad Hoc Network Routing Protocols: ARAN vs. endairA. In *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010,* José Luiz Fiadeiro, Stefania Gnesi, and Andrea Maggiolo-Schettini (Eds.). IEEE Computer Society, 191–202. https://doi.org/10.1109/SEFM.2010.24

[14] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *IMC'10.* ACM, 267–280.

[15] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. 1998. An Architecture for Differentiated Services. RFC 2475 (Informational). http://www.ietf.org/rfc/rfc2475.txt Updated by RFC 3260.

[16] Rakesh Bobba, Jon Grov, Indranil Gupta, Si Liu, José Meseguer, Peter Csaba Ölveczky, and Stephen Skeirik. 2018. Survivability: Design, Formal Modeling, and Validation of Cloud Storage Systems Using Maude. In *Assured Cloud Computing.* Wiley-IEEE Computer Society Press, Chapter 2, 10–48.

[17] R. Braden, D. Clark, and S. Shenker. 1994. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational). http://www.ietf.org/rfc/rfc1633.txt

[18] Bob Briscoe. 2007. Flow Rate Fairness: Dismantling a Religion. *SIGCOMM Comput. Commun. Rev.* 37, 2 (March 2007), 63–74. https://doi.org/10.1145/1232919.1232926

[19] CAIDA. 2021. Topology Research. https://www.caida.org/research/topology.

[20] Chen Chen, Limin Jia, Boon Thau Loo, and Wenchao Zhou. 2012. Reduction-based security analysis of Internet routing protocols. In *20th IEEE International Conference on Network Protocols, ICNP 2012, Austin, TX, USA, October 30 - Nov. 2, 2012.* IEEE Computer Society, 1–6. https://doi.org/10.1109/ICNP.2012.6459941

[21] Chen Chen, Limin Jia, Hao Xu, Cheng Luo, Wenchao Zhou, and Boon Thau Loo. 2015. A Program Logic for Verifying Secure Routing Protocols. *Logical Methods in Computer Science* 11, 4 (2015). https://doi.org/10.2168/LMCS-11(4:19)2015

[22] Rémy Chrétien and Stéphanie Delaune. 2013. Formal Analysis of Privacy for Routing Protocols in Mobile Ad Hoc Networks. In *Principles of Security and Trust - Second International Conference, POST 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7796),* David A. Basin and John C. Mitchell (Eds.). Springer, 1–20. https://doi.org/10.1007/978-3-642-36830-1_1

[23] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. 2007. *All About Maude.* LNCS, Vol. 4350. Springer.

[24] Véronique Cortier, Jan Degrieck, and Stéphanie Delaune. 2012. Analysing Routing Protocols: Four Nodes Topologies Are Sufficient. In *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012, Proceedings (Lecture Notes in Computer Science, Vol. 7215),* Pierpaolo Degano and Joshua D. Guttman (Eds.). Springer, 30–50. https://doi.org/10.1007/978-3-642-28641-4_3

[25] Yuri Gil Dantas, Vivek Nigam, and Iguatemi E. Fonseca. 2014. A Selective Defense for Application Layer DDoS Attacks. In *IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014, The Hague, The Netherlands, 24-26 September, 2014.* IEEE, 75–82. https://doi.org/10.1109/JISIC.2014.21

[26] A. Demers, S. Keshav, and S. Shenker. 1989. Analysis and Simulation of a Fair Queueing Algorithm. *ACM SIGCOMM Comp. Comm. Rev.* (1989).

[27] Tushar Deshpande, Panagiotis Katsaros, Stylianos Basagiannis, and Scott A. Smolka. 2011. Formal Analysis of the DNS Bandwidth Amplification Attack and Its Countermeasures Using Probabilistic Model Checking. In *13th IEEE International Symposium on High-Assurance Systems Engineering, HASE 2011, Boca Raton, FL, USA, November 10-12, 2011,* Taghi M. Khoshgoftaar (Ed.). IEEE Computer Society, 360–367. https://doi.org/10.1109/HASE.2011.57

[28] Adrian Farrel, Jean-Philippe Vasseur, and Jerry Ash. 2006. *A path computation element (PCE)-based architecture.* Technical Report.

[29] Seyed Kaveh Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd D. Millstein, Vyas Sekar, and George Varghese. 2016. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016,* Kimberly Keeton and Timothy Roscoe (Eds.). USENIX Association, 217–232. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/fayaz

[30] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. 2018. *Segment Routing Architecture.* Technical Report.

[31] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. 2015. A Coalgebraic Decision Procedure for NetKAT. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015,* Sriram K. Rajamani and David Walker (Eds.). ACM, 343–355. https://doi.org/10.1145/2676726.2677011

[32] Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin T. Vechev. 2018. Bayonet: probabilistic inference for networks. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018,* Jeffrey S. Foster and Dan Grossman (Eds.). ACM, 586–602. https://doi.org/10.1145/3192366.3192400

[33] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016,* Marinho P. Barcellos, Jon Crowcroft, Amin Vahdat, and Sachin Katti (Eds.). ACM, 300–313. https://doi.org/10.1145/2934872.2934876

[34] P Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. 2009. Pathlet Routing. In *ACM SIGCOMM Comp. Comm. Rev.*

[35] Bogdan Groza and Marius Minea. 2011. Formal modelling and automatic detection of resource exhaustion attacks. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2011, Hong Kong, China, March 22-24, 2011,* Bruce S. N. Cheung, Lucas Chi Kwong Hui, Ravi S. Sandhu, and Duncan S. Wong (Eds.). ACM, 326–333. https://doi.org/10.1145/1966913.1966955

[36] Garrett Hardin. 1968. The tragedy of the commons. *Science* (1968).

[37] Hsu-Chun Hsiao, Tiffany Hyun-Jin Kim, Soo Bum Lee, Xin Zhang, Sangjae Yoo, Virgil Gligor, and Adrian Perrig. 2013. STRIDE: Sanctuary Trail – Refuge from Internet DDoS Entrapment. In *AsiaCCS.*

[38] F. Baker J. Babiarz, K. Chan. [n.d.]. Configuration Guidelines for DiffServ Service Classes.

[39] Rahul Jain and Jean Walrand. 2010. An efficient Nash-implementation mechanism for network resource allocation. *Automatica* 46, 8 (2010), 1276–1283.

[40] Ramesh Johari and John N Tsitsiklis. 2009. Efficiency of scalar-parameterized mechanisms. *Operations Research* 57, 4 (2009), 823–839.

[41] Min Suk Kang and Virgil D. Gligor. 2014. Routing Bottlenecks in the Internet: Causes, Exploits, and Countermeasures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14).* ACM, New York, NY, USA, 321–333. https://doi.org/10.1145/2660267.

2660299
[42] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. 2013. The Crossfire Attack. In *IEEE S&P*.
[43] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, Steven D. Gribble and Dina Katabi (Eds.). USENIX Association, 113–126. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/kazemian
[44] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and Philip Brighten Godfrey. 2013. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, Nick Feamster and Jeffrey C. Mogul (Eds.). USENIX Association, 15–27. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/khurshid
[45] Dexter Kozen. 2014. NetKAT - A Formal System for the Verification of Networks. In *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8858)*, Jacques Garrigue (Ed.). Springer, 1–18. https://doi.org/10.1007/978-3-319-12736-1_1
[46] Soo Bum Lee and Virgil D. Gligor. 2010. FLoc: Dependable Link Access for Legitimate Traffic in Flooding Attacks. In *IEEE ICDCS*.
[47] Soo Bum Lee, Min Suk Kang, and Virgil D. Gligor. 2013. CoDef: Collaborative Defense against Large-Scale Link-Flooding Attacks. In *ACM CoNEXT*.
[48] M. Lepinski and S. Kent. 2012. An Infrastructure to Support Secure Internet Routing. RFC 6480 (Informational). http://www.ietf.org/rfc/rfc6480.txt
[49] Jed Liu, William T. Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Calin Cascaval, Nick McKeown, and Nate Foster. 2018. p4v: practical verification for programmable data planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, Sergey Gorinsky and János Tapolcai (Eds.). ACM, 490–503. https://doi.org/10.1145/3230543.3230582
[50] Si Liu, Peter Csaba Ölveczky, and José Meseguer. 2016. Modeling and analyzing mobile ad hoc networks in Real-Time Maude. *J. Log. Algebraic Methods Program.* 85, 1 (2016), 34–66. https://doi.org/10.1016/j.jlamp.2015.05.002
[51] Si Liu, Peter Csaba Ölveczky, Qi Wang, Indranil Gupta, and José Meseguer. 2019. Read atomic transactions with prevention of lost updates: ROLA and its formal analysis. *Formal Asp. Comput.* 31, 5 (2019), 503–540.
[52] Si Liu, Atul Sandur, José Meseguer, Peter Csaba Ölveczky, and Qi Wang. 2020. Generating Correct-by-Construction Distributed Implementations from Formal Maude Designs. In *NFM'20 (LNCS, Vol. 12229)*. Springer.
[53] Zhuotao Liu, Hao Jin, Yih-Chun Hu, and Michael Bailey. 2016. MiddlePolice: Toward enforcing destination-defined policies in the middle of the Internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1268–1279.
[54] Catherine A. Meadows. 2001. A Cost-Based Framework for Analysis of Denial of Service Networks. *Journal of Computer Security* 9, 1/2 (2001), 143–164. http://content.iospress.com/articles/journal-of-computer-security/jcs143
[55] José Meseguer. 1992. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theoretical Computer Science* 96, 1 (1992), 73–155.
[56] J. Nagle. 1985. On Packet Switches With Infinite Storage. RFC 970. http://www.ietf.org/rfc/rfc970.txt
[57] Sebastian Nanz and Chris Hankin. 2006. Formal Security Analysis for Ad-Hoc Networks. *Electr. Notes Theor. Comput. Sci.* 142 (2006), 195–213. https://doi.org/10.1016/j.entcs.2004.10.029
[58] Bryan Parno, Dan Wendlandt, Elaine Shi, Adrian Perrig, Bruce Maggs, and Yih-Chun Hu. 2007. Portcullis: Protecting Connection Setup from Denial-of-Capability Attacks. In *ACM SIGCOMM*.
[59] Adrian Perrig, Pawel Szalachowski, Raphael M Reischuk, and Laurent Chuat. 2017. *SCION: a secure internet architecture*. Springer.
[60] Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. 2016. Scaling network verification using symmetry and surgery. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 69–83. https://doi.org/10.1145/2837614.2837657
[61] Dominik Roos. 2018. *COLIBRI A Cooperative Lightweight Inter-domain Bandwidth Reservation Infrastructure*. Master's thesis. ETH Zurich, Zurich.
[62] Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2005. On Statistical Model Checking of Stochastic Systems. In *CAV (LNCS, Vol. 3576)*. Springer.
[63] Stanislav Shalunov and Benjamin Teitelbaum. 2003. 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?* (Karlsruhe, Germany) *(RIPQoS '03)*. ACM, New York, NY, USA, 137–140. https://doi.org/10.1145/944592.944600
[64] Ravinder Shankesi, Musab AlTurki, Ralf Sasse, Carl A. Gunter, and José Meseguer. 2009. Model-Checking DoS Amplification for VoIP Session Initiation. In *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5789)*, Michael Backes and Peng Ning (Eds.). Springer, 390–405. https://doi.org/10.1007/978-3-642-04444-1_24
[65] Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. 2019. Scalable verification of probabilistic networks. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, Kathryn S. McKinley and Kathleen Fisher (Eds.). ACM, 190–203. https://doi.org/10.1145/3314221.3314639
[66] Radu Stoenescu, Dragos Dumitrescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. 2018. Debugging P4 programs with Vera. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, Sergey Gorinsky and János Tapolcai (Eds.). ACM, 518–532. https://doi.org/10.1145/3230543.3230548
[67] Ahren Studer and Adrian Perrig. 2009. The Coremelt Attack. In *ESORICS*.
[68] Abraão Aires Urquiza, Musab A. AlTurki, Max I. Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, and Carolyn L. Talcott. 2019. Resource-Bounded Intruders in Denial of Service Attacks. In *CSF*. IEEE, 382–396. https://doi.org/10.1109/CSF.2019.00033
[69] US-CERT. 2017. Alert (TA17-164A) HIDDEN COBRA – North Korea's DDoS Botnet Infrastructure. https://www.us-cert.gov/ncas/alerts/TA17-164A.
[70] JP. Vasseur and JL. Le Roux. 2009. Path Computation Element (PCE) Communication Protocol (PCEP). RFC 5440 (Proposed Standard). http://www.ietf.org/rfc/rfc5440.txt Updated by RFC 7896.
[71] William Vickrey. 1961. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance* 16, 1 (1961), 8–37.
[72] Anduo Wang, Alexander J. T. Gurney, Xianglong Han, Jinyan Cao, Boon Thau Loo, Carolyn L. Talcott, and Andre Scedrov. 2014. A reduction-based approach towards scaling up formal analysis of internet configurations. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*. IEEE, 637–645. https://doi.org/10.1109/INFOCOM.2014.6847989
[73] Qi Wang, Pubali Datta, Wei Yang, Si Liu, Adam Bates, and Carl A. Gunter. 2019. Charting the Attack Surface of Trigger-Action IoT Platforms. In *CCS*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 1439–1453. https://doi.org/10.1145/3319535.3345662
[74] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. 2016. Scalable verification of border gateway protocol configurations with an SMT solver. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, Eelco Visser and Yannis Smaragdakis (Eds.). ACM, 765–780. https://doi.org/10.1145/2983990.2984012
[75] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *OSDI*. USENIX Association.
[76] Abraham Yaar, Adrian Perrig, and Dawn Song. 2004. SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks. In *IEEE S&P*.
[77] Xiaowei Yang, David Clark, and Arthur W Berger. 2007. NIRA: A New Inter-domain Routing Architecture. *IEEE/ACM Transactions on Networking* (2007).
[78] Xiaowei Yang, David Wetherall, and Thomas Anderson. 2005. A DoS-limiting Network Architecture. *ACM SIGCOMM Comp. Comm. Rev.* (2005).
[79] Yifei Yuan, Soo-Jin Moon, Sahil Uppal, Limin Jia, and Vyas Sekar. 2020. NetSMC: A Custom Symbolic Model Checker for Stateful Network Verification. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 181–200. https://www.usenix.org/conference/nsdi20/presentation/yuan
[80] Fuyuan Zhang, Limin Jia, Cristina Basescu, Tiffany Hyun-Jin Kim, Yih-Chun Hu, and Adrian Perrig. 2014. Mechanized Network Origin and Path Authenticity Proofs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 346–357. https://doi.org/10.1145/2660267.2660349
[81] Han Zhang, Chi Zhang, Arthur Azevedo de Amorim, Yuvraj Agarwal, Matt Fredrikson, and Limin Jia. 2021. Netter: Probabilistic, Stateful Network Models. In *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12597)*, Fritz Henglein, Sharon Shoham, and Yakir Vizel (Eds.). Springer, 486–508. https://doi.org/10.1007/978-3-030-67067-2_22
[82] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. 1993. RSVP: A New Resource ReSerVation Protocol. *IEEE Network* (1993).

A ADDITIONAL REQUIREMENTS

We also account for the following additional requirements:

(1) N-Tube should be *efficient* in computing bandwidth allocations by using only local information of the network, i.e., based on the demands of neighbor ASes;
(2) N-Tube should minimize its communication complexity by requiring only one round-trip per reservation request;
(3) N-Tube should be *scalable* by reducing AS administrators' configuration efforts. In particular, bandwidth allocations should be computed automatically;
(4) N-Tube should additionally allow administrators to specify bandwidth restrictions between adjacent ASes to adjust minimum bandwidth guarantees; and
(5) N-Tube should provide ASes with *flexibility* by allowing them to reserve segments of paths, and to update and delete reservations.

B ADDITIONAL INTUITION FOR MINIMAL BANDWIDTH GUARANTEE

To compute the request ratio $reqRatio_0$, we currently describe the simplest way, to take the ratio of the benign demand of `60 Gbps` and all adjusted demands *starting* from interface E. In Appendix D.5 we explain in detail how we divide demands on a link into two segments:

(1) The *starting segment* contains all reservations that start at this link and their adjusted demands are accumulated in *starting demands, startDem*; and
(2) The *transit segment* contains all reservations that traverse this link and their adjusted demands are accumulated in *transit demands transDem*.

Each of the two segments gets allocated a fixed proportion of the link's capacity. In Appendix D.5, we assume that this is exactly half the bandwidth, but this can be adapted as an additional parameter of N-Tube. By separating these two different kinds of demands into fixed bandwidth segments, we can guarantee that exceeding demands in one segment do not block out the allocations of the demands in the other segment. By providing a fixed proportion of the link's capacity for the transit segment and with the explanation in Section 3 we can provide a lower bound for the transit demands.

Note that there is still the possibility that a large number of (possibly malicious) ASes request excessive demands starting at a given link and therefore reduce the allocated bandwidth for benign demands in the starting segment. This due to the fact that the request ratio *reqRatio* of a request is computed as the ratio of the requested demand and accumulated adjusted demands given by *startDem*. Since the demands are adjusted and therefore upper-bounded by the links' capacities, the crucial factor is the *number* of requests starting at a given link.

A simple solution to avoid this reduction of benign demands is that each AS allows only a fixed number of other, e.g., trusted, ASes to start a reservation at its links. Alternatively, similarly as above, we can further split the starting segment into two sub-segments with a fixed portion of the segment's capacity:

(1) the *source segment* contains the *local requests* from the AS owning that link, i.e., the source AS of these requests, and
(2) the *external segment* contains the *telescoping requests* from external ASes that start their reservations at this link.

For each of these sub-segments, the request ratio of a reservation is computed as the ratio of the requested demand and the respective accumulated adjusted demands in this segment. Hence, as before between transit and starting demands, excessive demands in one of these two sub-segments cannot squeeze the allocated bandwidth in the other sub-segment.

The lower bound for the request ratio in the *source segment* is determined as the ratio of the requested demand and the starting segment's capacity multiplied by the number of end-hosts of source AS, which is exactly known to that AS. Possible excessive telescoping requests from external (possibly malicious) ASes starting at the link are isolated in the *external segment*, for which no fixed lower bound guarantees can be given.

In summary, for local reservation requests by honest ASes along a path starting from one of their own links, there is a lower bound for the requested ratio. Together with the local lower bounds at each AS on the path, we can provide a global lower bound *glb* for the whole path as described in Section 3.

C FORMAL PRELIMINARIES

C.1 Notation

As *base types*, $\mathbb{1} = \{\bot\}$ denotes the unit type, $\mathbb{B} = \{\text{TRUE}, \text{FALSE}\}$ denotes the booleans, $\mathbb{N}$ denotes the natural numbers, and $\mathbb{R}_0^+$ denotes the non-negative real numbers. For $a \leq b$, open and closed intervals are denoted by $]a;b[$ and $[a;b]$.

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 $A \rightharpoonup B = A \to B_\bot$ with $B_\bot = B + \mathbb{1}$, the *support* of a partial function g by $supp\,(g) = \{a \in A \mid g(a) \neq \bot\}$, and its *range* by $rng\,(g) = \{b \in B \mid \exists a \in A.\ g(a) = b\}$. We denote partial functions with finite support by $A \rightharpoonup_{fin} B$ and the undefined function with empty support by $\emptyset$. The updated function $f(x \mapsto y)$ is defined by $f(x \mapsto y)(x) = y$ and $f(x \mapsto y)(x') = y$ for all $x' \neq x$. A *record type* is a product type with named projections, e.g., $point = (\!|\, x \in \mathbb{N}; y \in \mathbb{N} \,|\!)$ with elements like $p = (\!|\, x = 1; y = 2 \,|\!)$ and fields $p.x$ and $p.y$. The term $p(\!|\, x := 3 \,|\!)$ denotes the updated point $(\!|\, x = 3; y = 2 \,|\!)$. The type $cpoint = point \oplus (\!|\, c \in color \,|\!)$ extends *point* with a *color* field.

The inductive type of *lists* over type A, denoted by $[A]$, is constructed from the empty list *nil* and the operation $a\#l$, which prepends an element $a \in A$ to a list $l \in [A]$. We write, e.g., $[1,2,3]$[1] for the list $1\#2\#3\#nil$, and $l[n]$ to retrieve the nth element of the list l, counting from 0.

We identify types with sets, and also use standard set notation. For example, for a set A we write $\mathbb{P}(A)$ for its *power set*, $\mathbb{P}_{fin}(A)$ for the set of its finite subsets, and $|A|$ for its *cardinality*.

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 $[1,3]$.

C.2 Labeled Transition Systems

A *labeled transition system* (LTS) $\mathfrak{T} = (\Sigma, \Sigma_0, \Lambda, \Delta)$ consists of a state space Σ, a set of initial states $\Sigma_0 \subseteq \Sigma$, a set of labels Λ, also called *events*, and a (labeled) transition relation $\Delta \in \Lambda \to \mathbb{P}(\Sigma \times \Sigma)$. *Executions* of $\mathfrak{T}$ 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 in an initial state, i.e., $\sigma_0 \in \Sigma_0$, and progresses according to the transition relation Δ, i.e., for all $n \in \mathbb{N}$, $(\sigma_n, \sigma_{n+1}) \in \Delta(\lambda_n)$.

To specify concrete models, we often use Λ-indexed families of *guards* $G_\lambda : \Sigma \to \mathbb{B}$ and *update* functions $U_\lambda : \Sigma \to \Sigma$. The induced transition relation is

$$\Delta(\lambda) = \{(\sigma, \sigma') \mid G_\lambda(\sigma) \ \wedge \ \sigma' = U_\lambda(\sigma)\}.$$

The relation $\sigma' = U_\lambda(\sigma)$ is called the *action* of the event. Moreover, we frequently use records as states and parametrized events, and we blur the distinction between an event λ and its associated state relation $\Delta(\lambda)$. For example, in the domain of banking, an event to withdraw an amount a of money from an account is specified by $withdraw(a) = \{(\sigma, \sigma') \mid \sigma.bal \geq a \ \wedge \ \sigma'.bal = \sigma.bal - a\}$. In this case, any state (record) field f that is not updated is implicitly left unchanged, e.g., $\sigma'.f = \sigma.f$.

D MODEL DETAILS

D.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 *tgt* to define a network:

Definition 1. Given three finite sets V, A, and I. We define a *network* $\eta \in \mathcal{N}$ as a record with

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

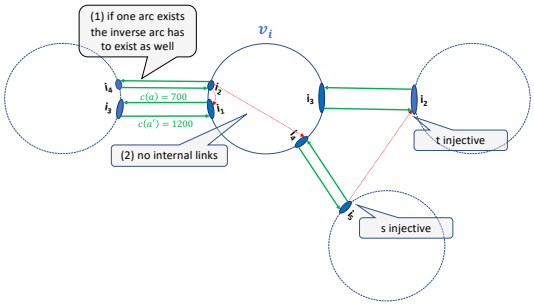
with the following components:

- 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/label 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 $sft : A \to V \times I$ and
  - ends in at exactly one interface of another AS given by the injective function $tgt : A \to V \times I$.
- To guarantee that G is a valid network graph the following constraints must hold:
  - No internal links

$$\forall u, v \in V, i, j \in I, a \in A : \\ sft(a) = (u, i) \wedge tgt(a) = (v, j) \Rightarrow u \neq v$$

  - Inverse links

$$\forall u, v \in V, i, j \in I, a \in A. \\ sft(a) = (u, i) \wedge tgt(a) = (v, j) \\ \Rightarrow \exists a' \in A.\ sft(a') = (v, j) \wedge tgt(a') = (u, i)$$



Definition 2. The set *environment* Γ is defined as

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

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.

D.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

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

- is loop-free

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

The set of ASes on a path p are defined by

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

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

D.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 $delM : \mathcal{M}_D \to \mathcal{M}$ and $resM : \mathcal{M}_R \to \mathcal{M}$.

For a valid message m the following must hold:

(1) Valid path, i.e., $m.path \in \mathcal{P}$

(2) Valid path counter

$$m.ptr \leq length(m.path)$$

(3) Valid pointers

$$m.first < m.last \leq length(m.path)$$

(4) Valid current bandwidth

$$length(m.accBW) = \max(0, m.ptr - m.first)$$

(5) Valid bandwidth range

$$m.minBW \leq m.maxBW \ \wedge \ 0 < m.maxBW$$

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

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

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

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

The function $nodes : \mathcal{M} \to \mathbb{P}(V)$ 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 $sgmt : \mathcal{M}_R \to \mathbb{P}(V)$ 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

$$m \sim m' :\Leftrightarrow$$
$$src(m) = src(m') \ \wedge \ m.id = m'.id \ \wedge \ m.idx = m'.idx$$

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

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

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

D.4 Reservation Maps

We model the *reservation maps* in the network as a partial function of type

$$ResMap = V \times V \times \mathbb{N} \rightharpoonup_{fin} Res.$$

A partial function $res \in ResMap$ stores for each AS x its *reservations* $res(x, s, id)$, 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 a record of type

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

containing a *version map* of the partial function type

$$\begin{aligned} VrsMap = \mathbb{N} \rightharpoonup_{fin} (\!| \, & minBW, maxBW, idBW, resBW \in \mathbb{R}_0^+; \\ & expT \in \mathbb{N} \, |\!). \end{aligned}$$

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

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

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

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

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 $sgmt : Res \to \mathbb{P}(V)$ 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

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

D.5 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, $resM'$ is obtained from $resM$ by removing the reservation corresponding to m. Second, $resM'_v$ is obtained by extracting the reservations that go through 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 $cap(x, e)$ 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.

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

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

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

Given the current time t, a *currently valid* version vrs is not *expired*, i.e., $vrs.expT \ge t$, and *successful*, i.e., $vrs.minBW \le vrs.resBW$. The reservation's bandwidth *allocation* are computed by the function *allocBW* and is defined as the maximum of its currently valid versions' $resBW$.[2]

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

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.

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 through 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*.

$$
\begin{aligned}
&ideal(m, resM, \delta, t) = \\
&\textbf{let} \\
&(\!| i, v, e |\!) = cur(m) \\
&vrs' = (\!| minBW := m.minBW; \\
&\qquad maxBW := m.maxBW; \\
&\qquad idBW := \min(\delta \cdot cap(v, i), m.maxBW, preIdBW(m)); \\
&\qquad resBW := m.minBW; \\
&\qquad expT := m.expT |\!) \\
&vrsM' = \emptyset\,(m.idx \mapsto vrs') \\
&res' = (\!| path := m.path; \\
&\qquad ptr := m.ptr; \\
&\qquad first := m.first; \\
&\qquad last := m.last; \\
&\qquad vrs := vrsM' |\!) \\
&resM' = resM\,((v, src(m), m.id) \mapsto res') \\
&resM'_v = filter(resM', v) \\
&tubeRatio = tubeRatio(v, i, e, resM'_v, t) \\
&\textbf{if } (m.first < m.ptr) \\
&\textbf{then } reqRatio = reqRatio_{transit}(v, src(m), m.id, i, resM'_v, t) \\
&\qquad linkRatio = linkRatio_{transit}(v, i, resM'_v, t) \\
&\textbf{else } reqRatio = reqRatio_{start}(v, src(m), m.id, i, resM'_v, t) \\
&\qquad linkRatio = linkRatio_{start}(v, i, resM'_v, t) \\
&\textbf{in} \\
&\min(\delta \cdot cap(v, i), m.maxBW, \\
&\qquad reqRatio \cdot linkRatio \cdot tubeRatio \cdot \delta \cdot cap(v, e)).
\end{aligned}
$$

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 $\min\{\delta \cdot cap(v, i), tubeDem(i, e)\}$, and the aggregated bounded tube demands at e.

$$
tubeRatio(v, i, e, resM, t) = \frac{\min\{\delta \cdot cap(v, i), tubeDem(v, i, e, resM, t)\}}{\sum_{i' \in I} \min\{\delta \cdot cap(v, i'), tubeDem(v, i', e, resM, t)\}}.
$$

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.

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

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

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

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*

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

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

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

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

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

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

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

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

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

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,

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

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

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

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*

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

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 is the sum of the previous *adjusted ideal bandwidth demands* of

traversing reservations,

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

Analogously, the *startDem* at an ingress interface i is the sum of the previous *adjusted ideal bandwidth demands* of starting reservations,

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

The (previous) *adjusted ideal bandwidth demand* of a reservation r is similarly defined to *adjReqDem*, the *adjusted requested demand*, with the *egress scaling factors* of the corresponding source AS and egress link

$$adjIdDem(v, r, resM, t) =$$
let
$$s = resSr(r)$$
$$i = resIn(r)$$
$$e = resEg(r)$$
in
$$egScalFctr(v, s, e, resM, t) \cdot \min\{\delta \cdot cap(v, i), \delta \cdot cap(v, e), idBW(r.vrs, t)\}.$$

Note, that we omit the ingress scaling factor *inScalFctr*, since the *previous AS* has already applied its egress scaling factor *egScalFctr*.

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*.

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

Request Ratio: If v is a transit AS on m's path, i.e., $m.first < m.ptr$ ($\leq m.last$), 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 define the *request ratio* by

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

D.6 State space

We define the set of *states* Σ as the record

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

A *state* $\sigma \in \Sigma$ describes a snapshot of the system at a given point in time, given by its *time* field. In our model, we assume discrete time, which is *loosely* synchronized between all ASes, 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 field *buf* of type $Buff = V \times I \to \mathbb{P}_{fin}(\mathcal{M})$ models network *buffers*, where $buf(x, i)$ holds the set of messages arrived at interface $i \in I$ of AS $x \in V$. The field *res* models all ASes' *reservation maps* as presented in Section 4.2 and formally defined in Appendix D.4. Finally, the field *kwl* models the *attackers' knowledge*: the set of messages created, collected, and shared by malicious ASes.

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

$$\sigma_0 = (\!| \; time := 0; buf := \emptyset; res := \emptyset; kwl := kwl_0 \; |\!),$$

where time starts at 0, the buffers and reservation map are initially empty, and the attackers' initial knowledge $kwl_0 = \{m \in \mathcal{M} \mid src(m) \in M\}$ consists of all messages with a malicious source AS.

D.7 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 in time to its successor state.

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

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.

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

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

$$
\begin{aligned}
&delRes(v, s \in V, id, idx \in \mathbb{N}, res \in ResMap) = \\
&\quad \textbf{let} \\
&\quad\ delVrs := res(v,s,id).vrs\,(idx \mapsto \bot) \\
&\quad \textbf{in} \\
&\quad\ res\,\big((v,s,id) \mapsto (\!|\ vrs := delVrs\ |\!)\big)
\end{aligned}
$$

D.8 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 the path is loop-free:

$$
\begin{aligned}
&PathCheck(p \in \mathcal{P}) = \\
&\forall k, k' \in \mathbb{N}, u, v \in V, i, e, i', e' \in I. \\
&\quad k < k' \leq length(p)\ \land\ p[k] = u_i^e\ \land\ p[k'] = v_{i'}^{e'}\ \Rightarrow\ u \neq v
\end{aligned}
$$

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

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

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

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

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

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

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:

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

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

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

$Rsvd : ResMap \times V \times \mathcal{M}_R \times \mathbb{N} \to \mathbb{B}$ 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.

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

The predicate *isMrkd* 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.

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

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.

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

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

Lemma 1.

$$
\begin{aligned}
isMrkd \land isRsvd &\Leftrightarrow \text{FALSE} \\
\neg(isMrkd \lor isRsvd) &\Leftrightarrow notRsvd
\end{aligned}
$$

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

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

$ResMapCheck : ResMap \times V \times \mathcal{M}_R \to \mathbb{B}$ 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:

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

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.,

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

This can be summarized in the property:

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

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

Lemma 2.

$$
\begin{aligned}
&ResMapCheck(res, v, m) \land isRsvd(res, v, m, t) \\
&\quad \Leftrightarrow isStrongRsvd(res, v, m, t)
\end{aligned}
$$

$ResMapCheck_D : ResMap \times V \times \mathcal{M}_D \to \mathbb{B}$ 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 $expT$.

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

D.9 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:

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

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.

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

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

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

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

D.10 Reservation Message Arrival Events

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

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

This event-template should be understood as follows:

- A reservation message m arrives at AS v at interface i at time $\sigma.time$ to be processed, i.e., $m \in \sigma.buf(v, i)$ and $\sigma.time = t$.
- The predicates *PathCheck* and *ResMsgCheck* ensure that the path $m.path$ and the m are well-formed. The predicate *ResMapCheck* ensures that the arriving reservation message m fits an already existing reservation with ID $m.id$ 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:

$$\begin{aligned}&FWD(\gamma, m, m', v, i, t) =\\ &\ RES(\gamma, m, m', v, i, t,\\ &\qquad atSrc \wedge \neg atSrt) \vee onWay, isFwd, notRsvd,\\ &\qquad nothing, forward, mark)\end{aligned}$$

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

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

The parameter *updBuf* is instantiated by the function *forward* which models the sending of the processed message m' forward along the path by removing the received message m from v's buffer at interface i and adding m'' to buffer i'' of the next AS v''.

$$\begin{aligned}&forward(v \in V, i \in I, buf \in Buff, m \in \mathcal{M}, m' \in \mathcal{M}) =\\ &\quad \textbf{let}\\ &\quad m'' = m'(\!| \ ptr := ptr + 1 \ |\!)\\ &\quad (\!| \ i'', v'', e'' \ |\!) = cur(m'')\\ &\quad \textbf{in}\\ &\quad buf \begin{pmatrix}(v,i) & \mapsto & buf(v,i) \setminus \{m\}\\ (v'',i'') & \mapsto & buf(v'',i'') \cup \{m''\}\end{pmatrix}\end{aligned}$$

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 $(src(m), m.id)$.

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

The function *preIdBW* extracts the ideal bandwidth computed by the previous AS from $m'.accBW$. In case there the message is at the first node, i.e., $m'.ptr = m'.first$, then the function returns $m'.maxBW$.

$$\begin{aligned}&preIdBW(m \in \mathcal{M}_R) =\\ &\quad \textbf{if } 0 \leq m.ptr - m.first - 1\\ &\quad \textbf{then } m.accBW[m.ptr - m.first - 1].idBW\\ &\quad \textbf{else } m.maxBW\end{aligned}$$

Note that $finBW$ is less or equal to $m'.maxBW$ and $\delta \cdot cap(v,i)$, since $m'.accBW$ contains the ideal bandwidth computed at AS v. However, the entries in $m'.accBW$ could exceed $m'.maxBW$ and $\delta \cdot cap(v,i)$, if the previous AS is malicious.

Computation event: This event is triggered by the arrival of a reservation message m at interface i of the start AS v at time $\sigma.time$ at the start and on the path (but not at the end) 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:

$$\begin{aligned}&CMP(\gamma, m, m', v, i, t) =\\ &\ RES(\gamma, m, m', v, i, t,\\ &\qquad (atSrt \wedge \neg atEnd) \vee onPth, isFwd, notRsvd,\\ &\qquad compute, forward, save)\end{aligned}$$

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

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

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 *compute*.

Turn event: This event is triggered by the arrival of a message m at interface i of AS v at time $\sigma.time$ at the end (but not the source) 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:

$$
\begin{aligned}
&TRN(\gamma, m, m', v, i, t) = \\
&RES(\gamma, m, m', v, i, t, \\
&\qquad atEnd \wedge \neg atSrc, isFwd, notRsvd, \\
&\qquad compute, backward, save)
\end{aligned}
$$

The parameter *updBuf* is initiated with the function *backward* which does the same as the function *forward* except sending m in the opposite direction along the path.

$$
\begin{aligned}
&backward(v \in V, i \in I, buf \in Buff, m \in \mathcal{M}, m' \in \mathcal{M}) = \\
&\quad \textbf{let} \\
&\quad m'' = m'(\!|\ ptr := ptr - 1\ |\!) \\
&\quad (\!|\ i'', v'', e''\ |\!) = cur(m'') \\
&\quad \textbf{in} \\
&\quad buf\begin{pmatrix}(v,i) & \mapsto & buf(v,i) \setminus \{m\} \\ (v'',i'') & \mapsto & buf(v'',i'') \cup \{m''\}\end{pmatrix}
\end{aligned}
$$

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

$$
\begin{aligned}
&UPT(\gamma, m, m', v, i, t) = \\
&RES(\gamma, m, m', v, i, t, \\
&\qquad onPth \vee (atSrt \wedge \neg atSrc), isBwd, isRsvd \vee isMrkd, \\
&\qquad nothing, backward, update)
\end{aligned}
$$

The parameter *updRes* is instantiated by the function *update*, which updates the entry derived from the information given by message m' in v's reservation map at entry $(src(m'), m'.id)$.

$$
\begin{aligned}
&update(v \in V, resM \in ResMap, m' \in \mathcal{M}_R) = \\
&\quad \textbf{let} \\
&\quad finBW = \min\left(m'.accBW\right) \\
&\quad res' = resM\left((v, src(m'), m'.id)\right) \\
&\quad vrsM' = res'.vrs \\
&\quad vrs' = vrsM'(m'.idx) \\
&\quad resBW'' = \min\left(vrs'.resBW, finBW\right) \\
&\quad vrs'' = vrs'(\!|\ resBW := resBW''\ |\!) \\
&\quad vrsM'' = vrsM'\left(m'.idx \mapsto vrs''\right) \\
&\quad res'' = res'(\!|\ vrs := vrsM''\ |\!) \\
&\quad \textbf{in} \\
&\quad resM\left((v, src(m'), m'.id) \mapsto res''\right).
\end{aligned}
$$

The field *resBW* is updated with the minimum of $m'.accBW$. Note, that $resBW'$ is limited by the previously computed *resBW* since there might be a malicious AS on the path which changed the values in *accBW* exceeding the links' capacities or *maxBW*. In contrast to *save* it is not guaranteed by the function *compute* that the last value in *accBW* is computed by the current AS.

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 $\sigma.time$. Using the event template *RES* it is instantiated[3]

$$
\begin{aligned}
&BWD(\gamma, m, m', v, i, t) = \\
&RES(\gamma, m, m', v, i, t, \\
&\qquad onWay, isBwd, isRsvd \vee isMrkd, \\
&\qquad nothing, backward, update).
\end{aligned}
$$

Finish event: This event is triggered by the arrival of a reservation message m at interface i of the source (but not at the end) AS v of the path at time $\sigma.time$, 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 by

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

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

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

One event: This event is triggered if the reservation message m only reserves at the source AS, i.e., $m.first = m.last = 0$, at interface i at time $\sigma.time$. The message completes its round-trip at the source,

[3]Note, that if the reservation's field *resBW* is undefined it holds

$$resBW' = \min(vrs'.resBW, m'.accBW) = \min(m'.accBW)$$

which computes the bandwidth, updates its reservation map *res* with the computed result, and drops the message. Using the event template *RES* it is instantiated by

$$\begin{aligned}&ONE(\gamma, m, m', v, i, t) = \\ &RES(\gamma, m, m', v, i, t, \\ &\quad atSrc \wedge atEnd, isFwd, isMrkd, \\ &\quad compute, drop, save).\end{aligned}$$

D.11 Deletion Events

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

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

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

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

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

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

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

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

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

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

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

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

D.12 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 $\sigma.time$

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

D.13 Attacker Events

Malicious ASes can execute two events: (i) receive a message, partially modify it, and store the resulting message in the attackers' knowledge *kwl*, and (ii) send a message in *kwl* to any neighbor AS in the network. Recall that *kwl* also includes any message in $\mathcal{M}$ with a malicious source AS. We now discuss these two events in more detail.

Collect event: In the event *CLT*, an 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*.

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

Here, the equivalence relation $m \approx m'$ expresses that m and m' coincide except on their mutable fields. This models our assumption that, in an N-Tube implementation, the source AS signs the immutable fields with its private key, while the mutable fields remain unprotected.

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

$$\begin{aligned} &ATK(m \in \mathcal{M}, a \in M, v \in H, i, e \in I, t \in \mathbb{N}) = \{(\sigma, \sigma') \mid \\ &\quad \texttt{- guards -} \\ &\quad m \in \sigma.kwl \ \wedge \ ((a, e), (v, i)) \in E \ \wedge \\ &\quad \texttt{- actions -} \\ &\quad \sigma'.buf = \sigma.buf((v, i) \mapsto \sigma.buf(v, i) \cup \{m\} \ \}. \end{aligned}$$

These two events model powerful attack capabilities. Malicious ASes can attack anytime, make arbitrary reservation requests from their own ASes, partially modify observed requests, replay old messages, and collude by communicating through out-of-band channels to share their knowledge and synchronize attacks. However, attackers cannot spoof messages from honest ASes, modify reservations stored in the reservation maps of honest ASes, or change the system's global time.

Strong attack event: The attack events are given as described above. However, when Theorem 1, 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.

D.14 Valid Executions

Monotonicity Given the transition relation Δ defined in Section 4.4 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

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

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 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.

Definition (Valid Executions)**.** An execution $\pi \in \mathfrak{E}$ is *valid* if it satisfies time progress (TP), message progress (MP), and (DP) properties.

This is satisfied if (i) all honest ASes run a fair scheduling algorithm (e.g., Round-Robin) to prevent message starvation, and (ii) messages are dropped in case of buffer overflow.

We will show that the global properties (G1–G5) of the N-Tube algorithm hold for all valid executions.

E PROPERTIES AND PROOFS

In this section, we define and prove the N-Tube's properties.

E.1 Successful Reservations and Constant Demands

Properties (G1) and (G2) assume that a *successful reservation* has been established by an honest source AS and properties (G3–G5) assume *constant demands*. We define these two notions in the following.

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

$$\begin{aligned}&\forall \hat{n}, n_c \in \mathbb{N}, \hat{v} \in V, \hat{i}, i \in I, \hat{m} \in \mathcal{M}_D, m_c \in \mathcal{M}_R, \hat{t}, t_c \in \mathbb{N}.\\ &\lambda_{n_c} = CRT_R(m_c, s, i, t_c) \wedge m_c \approx m \wedge\\ &\sigma_{\hat{n}}.time \in\ ]t_c; m.expT] \wedge\\ &\left(\lambda_{\hat{n}} = RMV(\hat{v}, \hat{i}, \hat{m}, \hat{t}) \ \vee\ \lambda_{\hat{n}} = DST(\hat{v}, \hat{i}, \hat{m}, \hat{t})\right)\\ &\quad \Rightarrow m_c \nsim \hat{m}.\end{aligned} \tag{nDE}$$

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

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

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

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

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

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

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

We say that a *state σ corresponds to D*, written $\sigma \vdash D$, if, for all honest ASes $v \in H$, the all reservations in res_v correspond to D at time $\sigma.time$, i.e.,

$$\forall v \in H, r \in rng(\sigma.res_v)\ .\ r, \sigma.time \vdash D$$

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 D between t_0 and t_1* if

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

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

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

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

- **Constant Demands:** any reservation confirmed by a reservation message m between t_0 and t_1, corresponds to D

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

and similarly for attack events

$$\begin{aligned}&\forall \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time \in [t_0, t_1] \Rightarrow\\ &\forall \tilde{m} \in \mathcal{M}_R, \tilde{i}, \tilde{e} \in I, \tilde{t} \in \mathbb{N}, \tilde{a} \in M, \tilde{v} \in sgmt(\tilde{m}).\\ &\lambda_{\tilde{n}} = ATK(\tilde{m}, \tilde{a}, \tilde{v}, \tilde{i}, \tilde{e}, \tilde{t})\\ &\quad \Rightarrow \tilde{m} \vdash D \ \wedge\ finBW(\tilde{m}) = D(src(\tilde{m}), \tilde{m}.id).maxBW\end{aligned}$$

- **No Deletion:** there are no deletion events between t_0 and t_1 for reservations given by $supp(D)$.

$$\begin{aligned}&\forall \tilde{n} \in \mathbb{N}, evt \in \{RMV, DST\}.\ \sigma_{\tilde{n}}.time \in [t_0, t_1]\\ &\forall \tilde{m} \in \mathcal{M}_D, \tilde{v} \in H, \tilde{i} \in I, \tilde{t} \in \mathbb{N}.\\ &\lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t}) \Rightarrow (src(\tilde{m}), \tilde{m}.id) \notin supp(D)\end{aligned}$$

E.2 Successful Reservation Theorem

Theorem 1 (Successful Reservation)**.** 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

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

Proof. First we prove that

$$\forall n \in \mathbb{N}, v \in V, k \in [m.first; m.last].$$
$$\sigma_n.time \in]t; m.expT] \ \wedge \ v = m.path[k].as$$
$$\Rightarrow \sigma_n.res(v, s, m.id).vrs(m.idx).resBW = finBW(m).$$

Given Assumption CF, we show that the *FIN* event is preceded by an *BWD* event on the previous AS. Given a message $m \in buf(v, i)$ 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 $\tilde{m}$

$$\forall v \in nodes(m).\ \exists evt \in \{CRT_R, FWD, CMP, TRN, UPT, BWD\}.$$
$$\exists \tilde{n} \in \mathbb{N}, \tilde{m}, \tilde{m}' \in \mathcal{M}_R, \tilde{i} \in I, \tilde{t} \in \mathbb{N}.$$
$$\sigma_{\tilde{n}}.time < t \wedge \lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', v, \tilde{i}, \tilde{t}) \wedge \tilde{m} \approx m$$

In the case distinction we need to exclude that the attacker put m into the buffer by using Lemma 8 which is based on our Assumption HP.

Next, we show that each of these events is unique for each successful reservation with message m

$$\forall \tilde{n}, \hat{n} \in \mathbb{N}, evt \in \{CRT_R, FWD, CMP, TRN, UPT, BWD\},$$
$$\forall \tilde{m}, \tilde{m}', \hat{m}, \hat{m}' \in \mathcal{M}_R, v \in H, \tilde{i}, \hat{i} \in I, \tilde{t}, \hat{t} \in \mathbb{N}.$$
$$\lambda_{\tilde{n}} = evt(\tilde{m}, \tilde{m}', v, \tilde{i}, \tilde{t}) \wedge \lambda_{\hat{n}} = evt(\hat{m}, \hat{m}', v, \hat{i}, \hat{t}) \wedge \ \tilde{m} \approx \hat{m}$$
$$\Rightarrow \tilde{n} = \hat{n}$$

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

$$\exists \tilde{n} < n, \tilde{m} \in \mathcal{M}_R.\ \tilde{m} \approx m \wedge$$
$$m.accBW[k - m.first] = (\!| \ avBW = avail(\tilde{m}, \sigma_{\tilde{n}}.res);$$
$$idBW = ideal(\tilde{m}, \sigma_{\tilde{n}}.res) \ |\!).$$

□

Consistency The following technical lemma shows that after a CRT_R event with message m the corresponding version is marked or reserved and remains unchanged until it expires.

Lemma 4 (CRT-isMrkd-until-expiration)**.**

$$\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in V, i \in I, t \in \mathbb{N}.$$
$$\lambda_n = CRT(m, v, i, t) \ \Rightarrow$$
$$\forall \hat{n} > n.\ \sigma_{\hat{n}}.time \leq m.expT \Rightarrow$$
$$(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)$$
$$\vee\ isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \wedge$$
$$ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

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 $\hat{n} > n$:

- $\hat{n} = n + 1$: Using assumption $\lambda_n = CRT(m, v, i, t)$ we can show that

  $$isMrkd(\sigma_{n+1}.res, v, m, \sigma_{n+1}.time)$$

  and

  $$ResMapCheck(\sigma_{n+1}.res, v, m).$$

  Hence, in this case ($\hat{n} = n + 1$) the claim holds.
- $\hat{n} \rightarrow \hat{n} + 1$: By *IH* it holds

  $$\hat{n} > n \ \wedge \ \sigma_{\hat{n}}.time \leq m.expT \Rightarrow$$
  $$(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)$$
  $$\vee\ isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \wedge$$
  $$ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

  We need to show:

  $$\hat{n} + 1 > n \ \wedge \ \sigma_{\hat{n}+1}.time \leq m.expT \Rightarrow$$
  $$(\ isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)$$
  $$\vee\ isRsvd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)\ ) \wedge$$
  $$ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)$$

  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:

    $$(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)$$
    $$\vee\ isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ ) \wedge$$
    $$ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

    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

    $$(\ isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}}.time)$$
    $$\vee\ isRsvd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}}.time)\ ) \wedge$$
    $$ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)$$

    By (9) of $ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)$ and assumption $\sigma_{\hat{n}+1}.time \leq m.expT$ it follows that

    $$(9')\ \sigma_{\hat{n}+1}(v, src(m), m.id).vrs(m.idx).expT$$
    $$=^{(9)} m.expT \geq \sigma_{\hat{n}+1}.time$$

    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:

$$(-)\ \big(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\ \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ \big) \land \\ ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

and applying $\sigma_{\hat{n}+1}.time = \sigma_{\hat{n}}.time$ again gives us:

$$(+)\ \big(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time) \\ \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time)\ \big) \land \\ ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

Regarding the reservation map there are two cases:

* $v = \tilde{v} \land src(m) = \tilde{s} \land m.id = \tilde{id} \land i\tilde{d}x = m.idx$: 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.

$$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\ = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)$$

and therefore the claim follows by (+).

– $CRT_R(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: As in RST it follows (+), hence in particular

$$\big(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\ \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ \big) \land \\ ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

The following cases:

* $\tilde{m} \approx m \land \tilde{m}.ptr = m.ptr \land \tilde{v} = v$: We can show that

$$isMrkd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time) \\ ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})$$

Together with $\tilde{m} \approx m$ and $\tilde{m}.ptr = m.ptr$ it follows that

$$isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time) \\ ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)$$

* $(\tilde{m} \not\approx m \lor \tilde{m}.ptr \neq m.ptr) \land \tilde{v} = v$: $\tilde{m} \not\approx m \lor \tilde{m}.ptr \neq m.ptr$ implies

$$(a)\ src(m) \neq src(\tilde{m}) \lor \\ (b)\ m.id \neq \tilde{m}.id \lor \\ (c)\ m.idx \neq \tilde{m}.idx \lor \\ (d)\ m.path \neq \tilde{m}.path \lor \\ (e)\ m.first \neq \tilde{m}.first \lor \\ (e)\ m.last \neq \tilde{m}.last \lor \\ (g)\ m.minBW \neq \tilde{m}.minBW \lor \\ (h)\ m.maxBW \neq \tilde{m}.maxBW \lor \\ (i)\ m.expT \neq \tilde{m}.expT \lor \\ (j)\ m.ptr \neq \tilde{m}.ptr$$

Two cases:

· $(a) \lor (b) \lor (c)$: The event's action do not affect the version of the reservation corresponding to m, i.e.

$$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx) \\ = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)$$

and the claim follows with (+).

· *Otherwise* : I.e.

$$src(m) = src(\tilde{m}) \land \\ m.id = \tilde{m}.id \land \\ m.idx = \tilde{m}.idx$$

but at least one of the other cases α in $(d), \ldots, (j)$ does not hold.
By (+) holds

$$\big(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \\ \lor isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)\ \big) \land \\ ResMapCheck(\sigma_{\hat{n}}.res, v, m)$$

hence

$$r = \sigma_{\hat{n}}.res(v, src(m), m.id) \land \\ \neg(0)\ r \neq \bot \land \\ (1)\ r.path = m.path \land \\ (2)\ r.ptr = m.ptr \land \\ (3)\ r.first = m.first \land \\ (4)\ r.last = m.last \land \\ \neg(5)\ r.vrs \neq \emptyset \\ \neg(6)\ r.vrs(m.idx) \neq \bot \land \\ (7)\ r.vrs(m.idx).minBW = m.minBW \land \\ (8)\ r.vrs(m.idx).maxBW = m.maxBW \land \\ (9)\ r.vrs(m.idx).expT = m.expT$$

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.

$\sigma_{\hat{n}}.res(v, src(m), m.id)$
$= \sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id) \wedge$
$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx)$
$= \sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id).vrs(\tilde{m}.idx) \wedge$
$\sigma_{\hat{n}}.res(v, src(m), m.id).\alpha = m.\alpha \wedge$
$\sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id).\alpha = \tilde{m}.\alpha \wedge$
$m.\alpha \neq \tilde{m}.\alpha$

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.

$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx)$
$= \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)$

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

$(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)$
$\vee \ isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \ ) \wedge$
$ResMapCheck(\sigma_{\hat{n}}.res, v, m)$

The following cases:

∗ $\tilde{m} \approx m \wedge \tilde{m}.ptr = m.ptr \wedge \tilde{v} = v$: We can show that

$isMrkd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time)$
$ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})$

Together with $\tilde{m} \approx m$ and $\tilde{m}.ptr = m.ptr$ it follows that

$isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)$
$ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)$

∗ $(\tilde{m} \not\approx m \vee \tilde{m}.ptr \neq m.ptr) \wedge \tilde{v} = v$: $\tilde{m} \not\approx m \vee \tilde{m}.ptr \neq m.ptr$ implies

$(a) \ src(m) \neq src(\tilde{m}) \vee$
$(b) \ m.id \neq \tilde{m}.id \vee$
$(c) \ m.idx \neq \tilde{m}.idx \vee$
$(d) \ m.path \neq \tilde{m}.path \vee$
$(e) \ m.first \neq \tilde{m}.first \vee$
$(e) \ m.last \neq \tilde{m}.last \vee$
$(g) \ m.minBW \neq \tilde{m}.minBW \vee$
$(h) \ m.maxBW \neq \tilde{m}.maxBW \vee$
$(i) \ m.expT \neq \tilde{m}.expT \vee$
$(j) \ m.ptr \neq \tilde{m}.ptr$

Two cases:

· $(a) \vee (b) \vee (c)$: The event's action do not affect the version of the reservation corresponding to m, i.e.

$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx)$
$= \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)$

and the claim follows with (+).

· *Otherwise* : I.e.

$src(m) = src(\tilde{m}) \wedge$
$m.id = \tilde{m}.id \wedge$
$m.idx = \tilde{m}.idx$

but at least one of the other cases α in $(d), \ldots, (j)$ does not hold.
By (+) holds

$(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time)$
$\vee \ isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}}.time) \ ) \wedge$
$ResMapCheck(\sigma_{\hat{n}}.res, v, m)$

hence

$r = \sigma_{\hat{n}}.res(v, src(m), m.id) \wedge$
$\neg(0) \ r \neq \bot \wedge$
$(1) \ r.path = m.path \wedge$
$(2) \ r.ptr = m.ptr \wedge$
$(3) \ r.first = m.first \wedge$
$(4) \ r.last = m.last \wedge$
$\neg(5) \ r.vrs \neq \emptyset$
$\neg(6) \ r.vrs(m.idx) \neq \bot \wedge$
$(7) \ r.vrs(m.idx).minBW = m.minBW \wedge$
$(8) \ r.vrs(m.idx).maxBW = m.maxBW \wedge$
$(9) \ r.vrs(m.idx).expT = m.expT$

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.

$\sigma_{\hat{n}}.res(v, src(m), m.id).\alpha = m.\alpha \wedge$
$\sigma_{\hat{n}}.res(v, src(\tilde{m}), \tilde{m}.id).\alpha = \tilde{m}.\alpha \wedge$
$m.\alpha \neq \tilde{m}.\alpha$

∗ $\tilde{v} \neq v$: The event's actions do not affect version of the reservation corresponding m, i.e.

$\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx)$
$= \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)$

and therefore the claim follows by (+).

– $CMP(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As in FWD with the following difference in case

$\tilde{m} \approx m \wedge \tilde{m}.ptr = m.ptr \wedge \tilde{v} = v$

We can show that

$isRsvd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time)$
$ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})$

instead of

$isMrkd(\sigma_{\hat{n}+1}.res, v, \tilde{m}, \sigma_{\hat{n}+1}.time)$
$ResMapCheck(\sigma_{\hat{n}+1}.res, v, \tilde{m})$

– $TRN(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: 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

$$\begin{aligned}&(\ isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time)\\ &\quad \vee isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time)\ ) \wedge\\ &ResMapCheck(\sigma_{\hat{n}}.res, v, m)\end{aligned}$$

  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 $\sigma_{\hat{n}}.res$ the same. Hence it holds

$$\begin{aligned}&isMrkd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time)\\ &\quad \Rightarrow isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)\\ &isRsvd(\sigma_{\hat{n}}.res, v, m, \sigma_{\hat{n}+1}.time)\\ &\quad \Rightarrow isMrkd(\sigma_{\hat{n}+1}.res, v, m, \sigma_{\hat{n}+1}.time)\\ &ResMapCheck(\sigma_{\hat{n}}.res, v, m)\\ &\quad \Rightarrow ResMapCheck(\sigma_{\hat{n}+1}.res, v, m)\end{aligned}$$

    and therefore the claim follows by (+).
  * *Otherwise* : The event's actions do not affect version of the reservation corresponding m, i.e.

$$\begin{aligned}&\sigma_{\hat{n}}.res(v, src(m), m.id).vrs(m.idx)\\ &\quad = \sigma_{\hat{n}+1}.res(v, src(m), m.id).vrs(m.idx)\end{aligned}$$

    and therefore the claim follows by (+).
- $DST(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{i}, \tilde{t})$: As in RMV.
- $DRP(\tilde{m}, \tilde{v}, \tilde{i}, \tilde{t})$: As in CRT_D.

□

Existence The following existence lemma shows that to any *FIN*-event at time t there was a *BWD* event before with an equivalent message $\bar{m}$ with the same *accBW* field at the previous AS $\bar{v}$ at time $\bar{t}$.

Lemma 5 (FIN-BWD-inductive-honest)**.**

$$\begin{aligned}&\forall k \in \mathbb{N}\\ &\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\ &\ \lambda_n = FIN(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\ &\ 0 \leq k < m.first\ \Rightarrow\\ &\ \exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\ &\quad \lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})\ \wedge\\ &\quad m \approx \bar{m}'\ \wedge\ \bar{m}'.ptr = k\ \wedge m.accBW = \bar{m}'.accBW\end{aligned}$$

Proof. Induction on k.

- $k = 0$: Let $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in H$, $e \in I$, $t \in \mathbb{N}$ with $\lambda_n = FIN(m, m', v, e, t)$ and $nodes(m) \subseteq H$. ($0 \leq k < m.first$ holds in this case) We can show that

$$\begin{aligned}&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\ &\ \lambda_n = FIN(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\ &\ 0 \leq k < m.first\ \Rightarrow\\ &\ \exists!\ \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\ &\quad \lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})\ \wedge\ m = \bar{m}'\end{aligned}$$

  From $m = \bar{m}'$, hence $m \approx \bar{m}'$. We can also show that $m.ptr = \bar{m}'.ptr = 0$, i.e. $k = 0 = \bar{m}'.ptr$, and $m.accBW = \bar{m}'.accBW$.
- $k \to k + 1$: By *IH* it holds

$$\begin{aligned}&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}.\\ &\ \lambda_n = FIN(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge\\ &\ 0 \leq k < m.first\ \Rightarrow\\ &\ \exists!\ \check{n} < n, \check{m}, \check{m}' \in \mathcal{M}_R, \check{v} \in H, \check{e} \in I, \check{t} \in \mathbb{N}.\\ &\quad \lambda_{\check{n}} = BWD(\check{m}, \check{m}', \check{v}, \check{e}, \check{t}) \wedge\\ &\quad m \approx \check{m}'\ \wedge\ \check{m}'.ptr = k\ \wedge m.accBW = \check{m}'.accBW\end{aligned}$$

  We need to show:

$$\begin{aligned}&\exists!\ \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\ &\ \lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t}) \wedge\\ &\ m \approx \bar{m}'\ \wedge\ \bar{m}'.ptr = k + 1\ \wedge m.accBW = \bar{m}'.accBW\end{aligned}$$

  Assume $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in H$, $e \in I$, $t \in \mathbb{N}$ with

$$\begin{aligned}&\lambda_n = FIN(m, m', v, e, t)\\ &nodes(m) \subseteq H\\ &0 \leq k + 1 < m.first.\end{aligned}$$

  From $0 \leq k + 1 < m.first$ it follows that $0 \leq k < m.first$ (Hence, by applying *IH* to k we get:

$$\begin{aligned}&(*)\ \exists \check{n} < n, \check{m}, \check{m}' \in \mathcal{M}_R, \check{v} \in H, \check{e} \in I, \check{t} \in \mathbb{N}.\\ &\quad \lambda_{\check{n}} = BWD(\check{m}, \check{m}', \check{v}, \check{e}, \check{t}) \wedge\\ &\quad m \approx \check{m}'\ \wedge\ \check{m}'.ptr = k\ \wedge m.accBW = \check{m}'.accBW\end{aligned}$$

  and therefore we have

$$\begin{aligned}&nodes(\check{m}) = nodes(m) \subseteq H\\ &\lambda_{\check{n}} = BWD(\check{m}, \check{m}', \check{v}, \check{e}, \check{t})\end{aligned}$$

  We then show that

$$\begin{aligned}&\exists \hat{n} < \check{n}, \hat{m}, \hat{m}' \in \mathcal{M}_R, \hat{v} \in H, \hat{e} \in I, \hat{t} \in \mathbb{N}.\\ &\quad \lambda_{\hat{n}} = BWD(\hat{v}, \hat{e}, \hat{m}, \hat{m}', \hat{t}) \wedge\\ &\quad \check{m} = \hat{m}'\end{aligned}$$

  Setting $\bar{n} := \hat{n}$ (i.e. $\bar{n} = \hat{n} < \check{n} < n$), $\bar{m} := \hat{m}$, $\bar{m}' := \hat{m}$, $\bar{v} := \hat{v}$, $\bar{e} := \hat{e}$, $\bar{t} := \hat{t}$ we get:

$$\begin{aligned}&\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}.\\ &\quad \lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})\end{aligned}$$

  By $\bar{m}' := \hat{m}'$, $\check{m} = \hat{m}'$, $\check{m} \approx \check{m}'$, and $m \approx \check{m}'$ we get:

$$m \approx \check{m}' \approx \check{m} = \hat{m}' = \bar{m}'$$

By $\bar{m}' := \hat{m}'$, $\check{m} = \hat{m}'$, $\check{m}.ptr = \check{m}'.ptr + 1$ (by BWD event), and $\check{m}'.ptr = k$ (by $(*)$) we get:

$$\bar{m}'.ptr = \hat{m}'.ptr = \check{m}.ptr = \check{m}'.ptr + 1 = k + 1$$

By $\bar{m}' := \hat{m}'$, $\check{m} = \hat{m}'$, $\check{m}.accBW = \check{m}'.accBW$ (by BWD event), and $m.accBW = \check{m}'.accBW$ (by $(*)$) we get:

$$\begin{aligned} m.accBW &= \check{m}'.accBW \\ &= \check{m}.accBW \\ &= \hat{m}'.accBW = \bar{m}'.accBW \end{aligned}$$

Together we get:

$$m \approx \bar{m}' \ \wedge \ \bar{m}'.ptr = k + 1 \ \wedge m.accBW = \bar{m}'.accBW.$$

□

Uniqueness Each message processing event with message m a creates a new version of the reservation corresponding to m. Using uniqueness lemmata like the following, we can inductively show that this can only happen at most once, since otherwise there is a contradiction with Lemma 4 given before. We provide as representative uniqueness lemma the the following one for event FIN.

Lemma 6 (FIN-uniqueness)**.**

$$\begin{aligned} &\forall n_1, n_2 \in \mathbb{N}, m_1, m_1', m_2, m_2' \in \mathcal{M}_R, v_1, v_2 \in V, e_1, e_2 \in I, t_1, t_2 \in \mathbb{N}. \\ &\lambda_{n_1} = FIN(m_1, m_1', v_1, e_1, t_1) \wedge \\ &\lambda_{n_2} = FIN(m_2, m_2', v_2, e_2, t_2) \wedge \\ &m_1 \sim m_2 \ \wedge \ v_1 = v_2 \wedge \\ &nodes(m_1) \subseteq H \wedge \\ &n_1 < n_2 \ \wedge \ \sigma_{n_2}.time \leq m_1.expT \\ &\Rightarrow n_1 = n_2 \end{aligned}$$

Proof. 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 $nodes(m_1) \subseteq H$ we can establish that

$$\begin{aligned} (1) \ \ &\exists \bar{n}_1 < n_1, \bar{m}_1, \bar{m}_1' \in \mathcal{M}_R, \bar{v}_1 \in H, \bar{e}_1 \in I, \bar{t}_1 \in \mathbb{N}. \\ &\lambda_{\bar{n}_1} = BWD(\bar{m}_1, \bar{m}_1', \bar{v}_1, \bar{e}_1, \bar{t}_1) \wedge \\ &m_1 = \bar{m}_1' \wedge \\ &\forall \hat{n}_1.\ \bar{n}_1 < \hat{n}_1 \leq n_1 \Rightarrow m_1 \in \sigma_{\hat{n}_1}.buf(v, e) \end{aligned}$$

and similarly with $m_1.path = m_2.path$ and $nodes(m_2) \subseteq H$ follows (2) for n_2.

$$\begin{aligned} (2) \ \ &\exists \bar{n}_2 < n_2, \bar{m}_2, \bar{m}_2' \in \mathcal{M}_R, \bar{v}_2 \in H, \bar{e}_2 \in I, \bar{t}_2 \in \mathbb{N}. \\ &\lambda_{\bar{n}_2} = BWD(\bar{m}_2, \bar{m}_2', \bar{v}_2, \bar{e}_2, \bar{t}_2) \wedge \\ &m_2 = \bar{m}_2' \wedge \\ &\forall \hat{n}_2.\ \bar{n}_2 < \hat{n}_2 \leq n_2 \Rightarrow m_2 \in \sigma_{\hat{n}_2}.buf(v, e) \end{aligned}$$

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:

$$\begin{aligned} nodes(\bar{m}_1) &=^{BWD(\bar{m}_1, \bar{m}_1', \ldots)} nodes(\bar{m}_1') \\ &=^{m_1 = \bar{m}_1'} nodes(m_1) \subseteq H \end{aligned}$$

and

$$\begin{aligned} \bar{m}_1.ptr &=^{BWD(\bar{m}_1, \bar{m}_1', \ldots)} \bar{m}_1'.ptr - 1 \\ &=^{m_1 = \bar{m}_1'} m_1.ptr - 1 \\ &=^{m_1.ptr = m_2.ptr} m_2.ptr - 1 \\ &=^{m_2 = \bar{m}_2'} \bar{m}_2'.ptr - 1 =^{BWD(\bar{m}_2, \bar{m}_2', \ldots)} \bar{m}_2.ptr \end{aligned}$$

and

$$\begin{aligned} \bar{m}_1.path &=^{BWD(\bar{m}_1, \bar{m}_1', \ldots)} \bar{m}_1'.path \\ &=^{m_1 = \bar{m}_1'} m_1.path \\ &=^{m_1.path = m_2.path} m_2.path \\ &=^{m_2 = \bar{m}_2'} \bar{m}_2'.path =^{BWD(\bar{m}_2, \bar{m}_2', \ldots)} \bar{m}_2.path \end{aligned}$$

Three cases:

- $\bar{n}_1 < \bar{n}_2$. Then it holds:

$$\begin{aligned} \sigma_{\bar{n}_2}.time &\leq^{\bar{n}_2 < n_2} \sigma_{n_2}.time \\ &\leq^{\sigma_{n_2}.time \leq m_1.expT} m_1.expT \\ &=^{m_1 = \bar{m}_1'} \bar{m}_1'.expT \\ &=^{BWD(\bar{m}_1, \bar{m}_1', \ldots)} \bar{m}_1.expT \end{aligned}$$

  From this we can show that $\bar{n} := \bar{n}_1 = \bar{n}_2$.
- $\bar{n}_2 < \bar{n}_1$: Then it holds:

$$\begin{aligned} \sigma_{\bar{n}_1}.time &\leq^{\bar{n}_1 < n_1} \sigma_{n_1}.time \\ &\leq^{n_1 < n_2} \sigma_{n_2}.time \\ &\leq^{\sigma_{n_2}.time \leq m_2.expT} m_2.expT \\ &=^{m_2 = \bar{m}_2'} \bar{m}_2'.expT \\ &=^{BWD(\bar{m}_2, \bar{m}_2', \ldots)} \bar{m}_2.expT \end{aligned}$$

  With this we can show that

$$\begin{aligned} &\lambda_{\bar{n}_1} = BWD(\bar{m}_1, \bar{m}_1', \bar{v}_1, \bar{e}_1, \bar{t}_1) \wedge \\ &\lambda_{\bar{n}_2} = BWD(\bar{m}_2, \bar{m}_2', \bar{v}_2, \bar{e}_2, \bar{t}_2) \wedge \\ &\bar{m}_1 \sim \bar{m}_2 \ \wedge \ \bar{m}_1.ptr = \bar{m}_2.ptr \ \wedge \ \bar{m}_1.path = \bar{m}_2.path \wedge \\ &nodes(\bar{m}_2) \subseteq H \wedge \\ &\bar{n}_2 < \bar{n}_1 \ \wedge \ \sigma_{\bar{n}_1}.time \leq \bar{m}_2.expT \end{aligned}$$

  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. □

Using existence lemmata we can show inductively that there was an UPT even for any AS on $sgmt(m)$ and using corresponding uniqueness lemmata we can show that there was exactly one.

Lemma 7 (FIN-UPT-inductive-honest)**.**

$$\begin{aligned}
&\forall k \in \mathbb{N} \\
&\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}. \\
&\ \lambda_n = FIN(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge \\
&\ m.first \leq k < m.last \implies \\
&\ \exists!\ \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}. \\
&\ \ \lambda_{\bar{n}} = UPT(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t}) \wedge \\
&\ \ m \approx \bar{m}' \ \wedge\ m.accBW = \bar{m}'.accBW
\end{aligned}$$

Proof. Induction on k.

- $k = m.first$: Let $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in H$, $e \in I$, $t \in \mathbb{N}$ with $\lambda_n = FIN(m, m', v, e, t)$ and $nodes(m) \subseteq H$ ($m.first \leq k < m.last$ holds in this case). We can show that

  $$\begin{aligned}
  (*)\ &\exists!\ \tilde{n} < n, \tilde{m}, \tilde{m}' \in \mathcal{M}_R, \tilde{v} \in H, \tilde{e} \in I, \tilde{t} \in \mathbb{N}. \\
  &\ \lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{e}, \tilde{t}) \wedge \\
  &\ m \approx \tilde{m}' \ \wedge\ \ \wedge m.accBW = \tilde{m}'.accBW
  \end{aligned}$$

  Given

  $$\begin{aligned}
  &\lambda_{\tilde{n}} = BWD(\tilde{m}, \tilde{m}', \tilde{v}, \tilde{e}, \tilde{t}) \\
  &nodes(\tilde{m}) = nodes(m) \subseteq H
  \end{aligned}$$

  we can then show that

  $$\begin{aligned}
  &\exists \bar{n} < \tilde{n}, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}. \\
  &\ \lambda_{\bar{n}} = UPT(\bar{v}, \bar{e}, \bar{m}, \bar{m}', \bar{t}) \wedge \\
  &\ \tilde{m} = \bar{m}'
  \end{aligned}$$

  By $\tilde{m} = \bar{m}'$, $\tilde{m} \approx \tilde{m}'$ (by BWD event) and $m \approx \tilde{m}'$ (by $(*)$) it follows

  $$m \approx \tilde{m}' \approx \tilde{m} = \bar{m}'$$

  By it follows:

  $$\bar{m}'.ptr = k$$

  By $\tilde{m} = \bar{m}'$, $\tilde{m}.accBW = \tilde{m}'.accBW$ (by BWD event) and $m.accBW \approx \tilde{m}'.accBW$ (by $(*)$) it follows

  $$m.accBW = \tilde{m}'.accBW = \tilde{m}.accBW = \bar{m}'.accBW$$

- $k \longrightarrow k + 1$: By *IH* it holds

  $$\begin{aligned}
  &\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, e \in I, t \in \mathbb{N}. \\
  &\ \lambda_n = FIN(m, m', v, e, t) \wedge nodes(m) \subseteq H \wedge \\
  &\ m.first \leq k < m.last \implies \\
  &\ \exists!\ \check{n} < n, \check{m}, \check{m}' \in \mathcal{M}_R, \check{v} \in H, \check{e} \in I, \check{t} \in \mathbb{N}. \\
  &\ \ \lambda_{\check{n}} = UPT(\check{m}, \check{m}', \check{v}, \check{e}, \check{t}) \wedge \\
  &\ \ m \approx \check{m}' \ \wedge\ \check{m}'.ptr = k \ \wedge m.accBW = \check{m}'.accBW
  \end{aligned}$$

  We need to show:

  $$\begin{aligned}
  &\exists!\ \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}. \\
  &\ \lambda_{\bar{n}} = UPT(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t}) \wedge \\
  &\ m \approx \bar{m}' \ \wedge\ \bar{m}'.ptr = k + 1 \ \wedge m.accBW = \bar{m}'.accBW
  \end{aligned}$$

  Assume $n \in \mathbb{N}$, $m, m' \in \mathcal{M}_R$, $v \in H$, $e \in I$, $t \in \mathbb{N}$ with

  $$\begin{aligned}
  &\lambda_n = FIN(m, m', v, e, t) \\
  &nodes(m) \subseteq H \\
  &m.first \leq k + 1 < m.last.
  \end{aligned}$$

  From $m.first \leq k + 1 < m.last$ it follows that $0 \leq k < m.last$ (
  Hence, by applying *IH* to k we get:

  $$\begin{aligned}
  (*)\ &\exists \check{n} < n, \check{m}, \check{m}' \in \mathcal{M}_R, \check{v} \in H, \check{e} \in I, \check{t} \in \mathbb{N}. \\
  &\ \lambda_{\check{n}} = UPT(\check{m}, \check{m}', \check{v}, \check{e}, \check{t}) \wedge \\
  &\ m \approx \check{m}' \ \wedge\ \check{m}'.ptr = k \ \wedge m.accBW = \check{m}'.accBW
  \end{aligned}$$

  and therefore we have

  $$\begin{aligned}
  &nodes(\check{m}) = nodes(m) \subseteq H \\
  &\lambda_{\check{n}} = UPT(\check{m}, \check{m}', \check{v}, \check{e}, \check{t}).
  \end{aligned}$$

  It then follows that

  $$\begin{aligned}
  &\exists \hat{n} < \check{n}, \hat{m}, \hat{m}' \in \mathcal{M}_R, \hat{v} \in H, \hat{e} \in I, \hat{t} \in \mathbb{N}. \\
  &\ \lambda_{\hat{n}} = UPT(\hat{v}, \hat{e}, \hat{m}, \hat{m}', \hat{t}) \wedge \\
  &\ \check{m} = \hat{m}'
  \end{aligned}$$

  Setting $\bar{n} := \hat{n}$ (i.e. $\bar{n} = \hat{n} < \check{n} < n$), $\bar{m} := \hat{m}$, $\bar{m}' := \hat{m}$, $\bar{v} := \hat{v}$, $\bar{e} := \hat{e}$, $\bar{t} := \hat{t}$ we get:

  $$\begin{aligned}
  &\exists \bar{n} < n, \bar{m}, \bar{m}' \in \mathcal{M}_R, \bar{v} \in H, \bar{e} \in I, \bar{t} \in \mathbb{N}. \\
  &\ \lambda_{\bar{n}} = BWD(\bar{m}, \bar{m}', \bar{v}, \bar{e}, \bar{t})
  \end{aligned}$$

  By $\bar{m}' := \hat{m}'$, $\check{m} = \hat{m}'$, $\check{m} \approx \check{m}'$, and $m \approx \check{m}'$ we get:

  $$m \approx \check{m}' \approx \check{m} = \hat{m}' = \bar{m}'$$

  By $\bar{m}' := \hat{m}'$, $\check{m} = \hat{m}'$, $\check{m}.ptr = \check{m}'.ptr + 1$ (by UPT event), and $\check{m}'.ptr = k$ (by $(*)$) we get:

  $$\bar{m}'.ptr = \hat{m}'.ptr = \check{m}.ptr = \check{m}'.ptr + 1 = k + 1$$

  By $\bar{m}' := \hat{m}'$, $\check{m} = \hat{m}'$, $\check{m}.accBW = \check{m}'.accBW$ (by UPT event), and $m.accBW = \check{m}'.accBW$ (by $(*)$) we get:

  $$\begin{aligned}
  m.accBW &= \check{m}'.accBW \\
  &= \check{m}.accBW \\
  &= \hat{m}'.accBW = \bar{m}'.accBW
  \end{aligned}$$

  Together we get:

  $$m \approx \bar{m}' \ \wedge\ \bar{m}'.ptr = k + 1 \ \wedge m.accBW = \bar{m}'.accBW$$

We can also show uniqueness, which establishes the lemma. □

Attacker Knowledge 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

Lemma 8 (Attacker-knowledge)**.**

$$\begin{aligned}
&\forall m \in \mathcal{M}_R. \\
&nodes(m) \subseteq H \implies \\
&\forall n \in \mathbb{N}, a \in M, i \in I.\ m \notin \sigma_n.kwl \ \wedge\ m \notin \sigma_n.buf(a, i).
\end{aligned}$$

Proof. We show by induction on $n \in \mathbb{N}$

$$\forall n \in \mathbb{N}, a \in M, i \in I, m \in \mathcal{M}_R.$$
$$nodes(m) \subseteq H \implies$$
$$m \notin \sigma_n.kwl \ \wedge \ m \notin \sigma_n.buf(a,i)$$

i.e. the induction hypothesis *IH* for σ_n is given by

$$\forall a \in M, i \in I, m \in \mathcal{M}_R.$$
$$(*)\ nodes(m) \subseteq H \implies$$
$$(1)\ m \notin \sigma_n.kwl \wedge$$
$$(2)\ m \notin \sigma_n.buf(a,i)$$

- $n = 0$: 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.
- $n \to n+1$: 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.

□

E.3 Stability Theorem

Theorem 2 (Stability Theorem)**.** If there are constant demands D between t_0 and t_1, then after time $t_0 + stabT$ all reservations allocate the ideal bandwidth until t_1, i.e.,

$$\exists \tilde{n} \in \mathbb{N}.\ \sigma_{\tilde{n}}.time = t_0 + stabT \wedge$$
$$\forall m \in rng(D), r \in Res, n > \tilde{n}, v \in sgmt(m).$$
$$\sigma_n.time \in]t_0 + stabT; t_1] \ \wedge \ r = \sigma_n.res_v(src(m), m.id)$$
$$\implies allocBW(r, \sigma_n.time) = \min_{x \in sgmt(m)}\{ideal(m, \sigma_{\tilde{n}}.res_x)\}$$

Proof. We first show that after $maxT$ the requested demands in all reservation maps correspond to D,

$$\forall n \in \mathbb{N}.\sigma_n.time \in]t_0 + maxT; t_1].\ \sigma_n \vdash D$$

From this follows that for any honest AS v and reservation r corresponding to a reservation message $m \in rng(D)$ the function $demBW$ remains constant and evaluates to $m.maxBW$

$$\forall n \in \mathbb{N}, v \in H, m \in rng(D), r \in Res.$$
$$\sigma_n.time \in]t_0 + maxT; t_1] \wedge r = \sigma_n.res_v(src(m), m.id)$$
$$\implies demBW(r,t) = m.maxBW.$$

Since the *ideal* bandwidth computation on the first AS of a path only depends on the value of the function $demBW_v$ and not on the *ideal* bandwidth computations executed at previous ASes on the path, it remains constant as well. We can show by induction on the length of the message's path that the *ideal* computations for all ASes on the path remain constant after time $t_0 + stabT$.

Using this, we show that the result of the *avail* computation is greater than that of the *ideal* computation for every renewal

$$\begin{aligned}&\forall n \in \mathbb{N}, evt \in \{CMP, UPT, TRN\}.\\ &\forall v \in H, m, m' \in \mathcal{M}_R, i \in I, t \in \mathbb{N}\\ &\sigma_n.time \in]t_0 + stabT; t_1] \wedge \lambda_n = evt(m, m', v, i, t) \wedge m \vdash D\\ &\quad \Rightarrow ideal(m, \sigma_n.res_v) \geq avail(m, \sigma_n.res_v).\end{aligned}$$

Hence, together with the definition of *finBW*

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

it follows that

$$allocBW\ (r, \sigma_n.time) = \min_{x \in sgmt(m)}\{ideal(m, \sigma_{\tilde{n}}.res_x)\}$$

as required. □

E.4 Local Properties of N-Tube's Computation

For a valid reservation message m, N-Tube's bandwidth allocation computation has the following four *local* properties: positivity, lower ideal bound, bounded-tube proportionality, and per-request proportionality. We define and prove these properties below. These properties hold at each AS, and are later used to prove the global properties (G1–G5).

As illustrated in Section 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.

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

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

Analogous definitions apply to ingress links.

Positivity: The functions *avail* and *ideal* always compute strictly positive values.

Lemma 9 (Positivity)**.** For a valid request with $(*)$ $m.minBW = 0$ and $(+)$ $nodes(m) \subseteq H$ a positive amount of bandwidth is always allocated:

$$\begin{aligned}&\forall event \in \{CMP, TRN, UPT\}.\\ &\forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N}.\\ &\lambda_n = event(m, m', v, i, t) \Rightarrow finBW(m, resM_v) > 0.\end{aligned}$$

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

(1) $m.maxBW > 0$,
(2) $cap(v, e) > 0$,
(3) $cap(v, i) > 0$.

Furthermore, by the event's action it holds that

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

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) $r.first < r.ptr < r.last$
(5) $r.vrs(m.idx).expT > t$

By the function *compute*

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

it follows that

$$\begin{aligned}&m'.accBW[m'.ptr] =\\ &(\!|\ avBW := avail(m, res, \delta, t); idBW := ideal(m, res, \delta, t)\ |\!)\end{aligned}$$

Since $m'.maxBW = m.maxBW$ and (1) it suffices to show that both *avail* and *ideal* return a positive values.

– *avail* : By definition

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

By Lemma 10 it follows that

$$cap(v,e) > \sum_{\substack{r \in rng(resM'_v):\\ resEg(r)=e}} allocBW(r.vrs, t)$$

Note, that in Lemma 10 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 $reqRatio_{start}$)

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

is given by

$$adjIdDem(v, r, resM, t) = egScalFctr(v, s, e, resM, t) \cdot \min\{cap(v,i), cap(v,e), idBW(r.vrs, t)\}$$

which by (2) and (3) is positive if *egScalFctr* and *idBW* are positive.

∗ *egScalFctr* : By the definition of *egScalFctr*

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

and assumption (2) it suffices to show that $egDem(v, s, e, resM, t)$ is positive. By the definition of *egDem*

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

it suffices to show that $reqDem(v, r, i, e, t)$ is positive. By the definition of *reqDem*

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

and (2) and (3) it suffices to show that $demBW(r.vrs, t)$ is positive. By definition of *demBW*

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

it suffices to show that

(a) $r.vrs(m'.idx).maxBW > 0$

(b) $r.vrs(m'.idx).minBW \le r.vrs(m'.idx).resBW$

(c) $r.vrs(m'.idx).expT \ge t$

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*

$$idBW(vrsM, t) = \max_{\substack{vrs \in\\ rng(vrsM)}} \{vrs.idBW \mid vrs.minBW \le vrs.resBW \wedge vrs.expT \ge t\}.$$

it suffices to show that

(a) $r.vrs(m'.idx).idBW > 0$

(b) $r.vrs(m'.idx).minBW \le r.vrs(m'.idx).resBW$

(c) $r.vrs(m'.idx).expT \ge t$

The fact (a) follows by the definition of the function *save*, in particular by

$$r.vrs(m'.idx).idBW := m'.accBW.[m'.ptr - 1].idBW,$$

We can show using assumption (+) that 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

$$ideal(m, resM, \delta, t) \le \delta \cdot cap(v,e).$$

□

Lemma 10 (Capacity-constraint-invariant)**.**

$$\forall n \in \mathbb{N}, s, v \in H, e \in I, id \in \mathbb{N}, t \in \mathbb{N}.$$
$$resM'_v = filter\,(\sigma_n.res(v, s, id), v) \;\wedge\; cap(v, e) > 0$$
$$\Rightarrow cap(v, e) > \sum_{\substack{r \in rng(resM'_v):\\ resEg(r)=e}} allocBW(r.vrs, t)$$

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 $r \in rng\,(resM'_v)$. 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

  $$resM'_v = filter\,(\sigma_n(v, s, id).res, v) \wedge cap(v, e) > 0$$

  By *IH*

  $$\forall s, v \in H, e \in I, id \in \mathbb{N}, t \in \mathbb{N}.$$
  $$resM'_v = filter\,(\sigma_n(v, s, id).res, v) \wedge cap(v, e) > 0$$
  $$\Rightarrow cap(v, e) > \sum_{\substack{r \in rng(resM'_v):\\ resEg(r)=e}} allocBW(r.vrs, t)$$

  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*

  $$filter(resM, v) =$$
  $$\lambda(s', id').$$
  $$\textbf{let } r = resM(v, s', id')$$
  $$\textbf{in } \left(\textbf{if } r.first \leq r.ptr \leq r.last \textbf{ then } resM(v, s', id') \textbf{ else } \bot\right)$$

  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

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

  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) $r.first < r.ptr < r.last$

  (5) $r.vrs(\tilde{m}.idx).expT > t$

  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

  $$resM'_v = filter\,(\sigma_{n+1}(v, s, id).res, v) \wedge cap(v, e) > 0$$
  $$\Rightarrow cap(v, e) > \sum_{\substack{\tilde{r} \in rng(resM'_v):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)$$

  The only reservation that changed from σ_n to σ_{n+1} is r to r', hence it holds (a)

  $$\sum_{\substack{\tilde{r} \in rng(resM'_v):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)$$
  $$= \sum_{\substack{\tilde{r} \in rng(resM'_v):\\ \tilde{r} \neq r' resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})$$
  $$= \sum_{\substack{\tilde{r} \in rng(resM_v):\\ \tilde{r} \neq r resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})$$

  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*

    $$\sum_{\substack{\tilde{r} \in rng(resM'_v):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)$$
    $$=^{(c)} \sum_{\substack{\tilde{r} \in rng(resM_v):\\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t})$$
    $$=^{(d)} \sum_{\substack{\tilde{r} \in rng(resM_v):\\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r.vrs, \tilde{t})$$
    $$= \sum_{\substack{\tilde{r} \in rng(resM_v):\\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t)$$
    $$<^{IH} cap(v, e)$$

    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

$$\begin{aligned} & r'.vrs(\tilde{m}.idx).resBW \\ & \quad := \min(\tilde{m}'.maxBW, \min(\tilde{m}'.accBW)) \\ & \quad \leq \tilde{m}'.accBW[\tilde{m}'.ptr].avBW \\ & \quad = avail(\tilde{m}, resM_v, \delta, \tilde{t}) \end{aligned}$$

hence, altogether it holds

$$(e)\ allocBW(r'.vrs, \tilde{t}) \leq avail(\tilde{m}, resM_v, \delta, \tilde{t})$$

By the definition of *avail*

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

Applying this to (c) and

$$\begin{aligned} & \sum_{\substack{\tilde{r} \in rng(resM'_v): \\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \\ & =^{(c)} \sum_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + allocBW(r'.vrs, \tilde{t}) \\ & =^{(e)} \sum_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + \\ & \quad avail(\tilde{m}, resM_v, \delta, \tilde{t}) \\ & =^{(f)} \sum_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) + \\ & \quad \delta \left(cap(v,e) - \sum_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \right) \\ & = \delta cap(v,e) + (1-\delta) \sum_{\substack{\tilde{r} \in rng(resM_v): \\ \tilde{r} \neq r \wedge resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \\ & \leq \delta cap(v,e) + (1-\delta) \sum_{\substack{\tilde{r} \in rng(resM_v): \\ resEg(\tilde{r})=e}} allocBW(\tilde{r}.vrs, t) \\ & <^{IH} \delta cap(v,e) + (1-\delta) cap(v,e) = cap(v,e) \end{aligned}$$

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.

□

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.

$$\begin{aligned} & \forall event \in \{CMP, TRN, UPT\}. \\ & \forall n \in \mathbb{N}, m, m' \in \mathcal{M}_R, v \in H, i \in I, t \in \mathbb{N}. \\ & \lambda_n = event(m, m', v, i, t) \Rightarrow \\ & \exists G > 0. \forall x \in sgmt(m) \cap H.\ ideal(m, resM_x) > G \cdot r_v \end{aligned}$$

Lemma 11 (Ideal-computation-at-AS)**.**

$$\begin{aligned} & \forall m \in \mathcal{M}_R, s \in H, v \in sgmt(m) \setminus \{first(m)\} \cap H, i, e \in I, resM \in ResMap, t \in \mathbb{N}. \\ & cur(m) = v_i^e \ \wedge\ s = src(m) \ \wedge\ vrs = resM(v, src(m), m.id).vrs(m.idx) \wedge \\ & 0 < vrs.maxBW = m.maxBW \wedge \\ & m.accBW[m.first].idBW = m.maxBW \wedge \\ & vrs.idBW = m.accBW[m.ptr].idBW \wedge \\ & egDem(v, src(m), e, resM, t) \leq cap(v,e) \wedge \\ & inDem(v, src(m), i, resM, t) \leq cap(v,i) \wedge \\ & transitDem(v, i, resM, t) \leq cap(v,i) \\ & \Rightarrow \exists G_v > 0.\ ideal(m, resM, \delta, t) \geq G_v \cdot m.accBW[m.ptr].idBW \end{aligned}$$

and for $v = first(m) \cap H$

$$\begin{aligned} & \exists G_v > 0.\ ideal(m, resM, \delta, t) \geq \\ & \qquad G_v \cdot reqRatio_{start}(v, s, id, i, resM, t) \cdot m.maxBW \end{aligned}$$

with

$$G_v := \frac{m.maxBW}{2 \cdot cap(v,i) \cdot \sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e).$$

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

$$(a)\ \ 0 < m.maxBW$$
$$(b)\ \ egDem(v, src(m), e, resM, t) \leq cap(v, e)$$
$$(c)\ \ inDem(v, src(m), i, resM, t) \leq cap(v, i)$$
$$(d)\ \ transitDem(v, i, resM, t) \leq cap(v, i)$$
$$(e)\ \ 0 < vrs.maxBW = m.maxBW$$
$$(f)\ \ vrs.idBW = m.accBW[m.ptr].idBW$$
$$(g)\ \ m.accBW[m.first].idBW = m.maxBW$$

By the definition of *ideal*

$$\begin{aligned}
&ideal(m, resM, \delta, t) = \\
&\textbf{let} \\
&(\!| i, v, e |\!) = cur(m) \\
&vrs' = (\!| minBW := m.minBW; \\
&\qquad maxBW := m.maxBW; \\
&\qquad idBW := \min(\delta \cdot cap(v, i), m.maxBW, preIdBW(m)); \\
&\qquad resBW := m.minBW; \\
&\qquad expT := m.expT |\!) \\
&vrsM' = \emptyset\,(m.idx \mapsto vrs') \\
&res' = (\!| path := m.path; \\
&\qquad ptr := m.ptr; \\
&\qquad first := m.first; \\
&\qquad last := m.last; \\
&\qquad vrs := vrsM' |\!) \\
&resM' = resM\,((v, src(m), m.id) \mapsto res') \\
&resM'_v = filter(resM', v) \\
&tubeRatio = tubeRatio(v, i, e, resM'_v, t) \\
&\textbf{if } (m.first < m.ptr) \\
&\textbf{then } reqRatio = reqRatio_{transit}(v, src(m), m.id, i, resM'_v, t) \\
&\qquad linkRatio = linkRatio_{transit}(v, i, resM'_v, t) \\
&\textbf{else } reqRatio = reqRatio_{start}(v, src(m), m.id, i, resM'_v, t) \\
&\qquad linkRatio = linkRatio_{start}(v, i, resM'_v, t) \\
&\textbf{in} \\
&\min(\delta \cdot cap(v, i), m.maxBW, \\
&\qquad reqRatio \cdot linkRatio \cdot tubeRatio \cdot \delta \cdot cap(v, e)).
\end{aligned}$$

we consider the term:

$$reqRatio \cdot linkRatio \cdot tubeRatio(v, i, e, resM'_v, t) \cdot \delta \cdot cap(v, e)$$

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*

$$\begin{aligned}
&egDem(v, s, e, resM, t) = \\
&\sum_{\substack{r' \in rng(resM): \\ resSr(r')=s \\ resEg(r')=e}} reqDem(v, r', resIn(r'), e, t).r.path[r.ptr].inI, e).
\end{aligned}$$

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*

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

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 $v = first(m)$ and $v \in sgmt(m) \setminus first(m)$. By the definition of $linkRatio_{transit}$

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

By (d) it follows for its nominator

$$
\begin{aligned}
(D)\ &\min\{cap(v,i), trDem\} \\
&= \min\{cap(v,i), transitDem(v,i,resM,t)\} \\
&=^{(d)} transitDem(v,i,resM,t)
\end{aligned}
$$

and therefore

$$
\begin{aligned}
&linkRatio_{transit}(v,i,resM,t) \\
&= \frac{\min\{cap(v,i), trDem\}}{\min\{cap(v,i), stDem\} + \min\{cap(v,i), trDem\}} \\
&\geq \frac{\min\{cap(v,i), trDem\}}{2 \cdot cap(v,i)} \\
&=^{(D)} \frac{transitDem(v,i,resM,t)}{2 \cdot cap(v,i)}
\end{aligned}
$$

- $linkRatio_{start}$: In this case (D) does not hold, but by (C) and (g) it follows

$$
\begin{aligned}
(E)\ &adjIdDem(v,r,resM,t) \\
&\geq^{(C)} m.accBW[m.ptr].idBW \\
&=^{(g)} m.maxBW
\end{aligned}
$$

and we obtain as lower bound

$$
\begin{aligned}
&linkRatio_{start}(v,i,resM,t) \\
&= \frac{\min\{cap(v,i), stDem\}}{\min\{cap(v,i), stDem\} + \min\{cap(v,i), trDem\}} \\
&\geq \frac{\min\{cap(v,i), stDem\}}{2 \cdot cap(v,i)} \\
&\geq^{(E)} \frac{m.maxBW}{2 \cdot cap(v,i)}
\end{aligned}
$$

- $reqRatio_{transit}$: By the definition of *adjIdDem*

$$
\begin{aligned}
&adjIdDem(v,r,resM,t) = \\
&egScalFctr(v,s,e,resM,t) \cdot \min\{cap(v,i), cap(v,e), idBW(r.vrs,t)\}.
\end{aligned}
$$

and similar as above $(*)$ $egScalFctr(v,s,e,resM,t) = 1$ and $(+)$ $idBW(r.vrs,t) \leq cap(v,i), cap(v,e)$ it follows that

$$
\begin{aligned}
(C)\ &adjIdDem(v,r,resM,t) \\
&=^{(+),(*)} idBW(r.vrs,t) \\
&\geq r.vrs(m.idx).idBW \\
&=^{(f)} m.accBW[m.ptr].idBW
\end{aligned}
$$

By (C) and (d) and the definition of $reqRatio_{transit}$

$$
\begin{aligned}
&reqRatio_{transit}(v,s,id,i,resM,t) \\
&= \frac{adjIdDem(v, resM(v,s,id), resM, t)}{transitDem(v,i,resM,t)}
\end{aligned}
$$

it follows

$$
\begin{aligned}
&reqRatio_{transit}(v,s,id,i,resM,t) \\
&= \frac{adjIdDem(v, resM(v,s,id), resM, t)}{transitDem(v,i,resM,t)} \\
&\geq^{(C)} \frac{m.accBW[m.ptr].idBW}{transitDem(v,i,resM,t)}
\end{aligned}
$$

- $reqRatio_{start}$: By (E) and the definition of $reqRatio_{start}$ it follows

$$
\begin{aligned}
&reqRatio_{start}(v,s,id,i,resM,t) \\
&= \frac{adjIdDem(v, resM(v,s,id), resM, t)}{startDem(v,i,resM,t)} \\
&\geq^{(E)} \frac{m.maxBW}{startDem(v,i,resM,t)}
\end{aligned}
$$

Altogether we obtain the following lower bounds for *ideal* depending on the two cases $v = first(m)$ and $v \in sgmt(m) \setminus \{first(m)\}$, respectively:

- $v = first(m)$:

$$
\begin{aligned}
&ideal(m,resM,\delta,t) \\
&= reqRatio_{start} \cdot linkRatio_{start} \cdot tubeRatio \cdot \delta \cdot cap(v,e) \\
&\geq reqRatio_{start} \cdot \frac{m.maxBW}{2 \cdot cap(v,i)} \cdot \frac{m.maxBW}{\sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e) \\
&= reqRatio_{start} \cdot m.maxBW \cdot \frac{m.maxBW}{2 \cdot cap(v,i) \cdot \sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e)
\end{aligned}
$$

Hence, we can set

$$
G_v := \frac{m.maxBW}{2 \cdot cap(v,i) \cdot \sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e)
$$

which only depends on $m.maxBW$ and the capacities on $m.path$ of the network.

- $v \in sgmt(m) \setminus \{first(m)\}$:

$$
\begin{aligned}
&ideal(m,resM,\delta,t) \\
&= reqRatio_{transit} \cdot linkRatio_{transit} \cdot tubeRatio \cdot \delta \cdot cap(v,e) \\
&\geq \frac{m.accBW[m.ptr].idBW}{trDem} \cdot \frac{trDem}{2 \cdot cap(v,i)} \cdot \frac{m.maxBW}{\sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e) \\
&= m.accBW[m.ptr].idBW \cdot \frac{m.maxBW}{2 \cdot cap(v,i) \cdot \sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e)
\end{aligned}
$$

Hence, we can set

$$
G_v := \frac{m.maxBW}{2 \cdot cap(v,i) \cdot \sum_{i' \in I_v} cap(v,i')} \cdot \delta \cdot cap(v,e)
$$

which only depends on $m.maxBW$ and the capacities on $m.path$ of the network.

□

Lemma 12 (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.,

$$(a)\ \ \sum_{\tilde{s}\in V} inDem(x,\tilde{s},i,resM,t) \le cap(x,i)$$
$$(b)\ \ \sum_{\tilde{s}\in V} inDem(x,\tilde{s},i',resM,t) \le cap(x,i').$$

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

$$\begin{aligned}
&\frac{tubeRatio(x,i,e,resM,t)}{tubeRatio(x,i',e,resM,t)}\\
&= \frac{\frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}}{\frac{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}}\\
&= \frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}}\\
&=^{(a'),(b')} \frac{tubeDem(x,i,e,resM,t)}{tubeDem(x,i',e,resM,t)}
\end{aligned}$$

Independent from ingress link i' being congestion, if i is not congested then it follows

$$\begin{aligned}
&\frac{tubeRatio(x,i,e,resM,t)}{tubeRatio(x,i',e,resM,t)}\\
&= \frac{\frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}}{\frac{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}}{\sum_{\tilde{i}\in I}\min\{cap(x,\tilde{i}),tubeDem(x,\tilde{i},e,resM,t)\}}}\\
&= \frac{\min\{cap(x,i),tubeDem(x,i,e,resM,t)\}}{\min\{cap(x,i'),tubeDem(x,i',e,resM,t)\}}\\
&\ge^{(a')} \frac{tubeDem(x,i,e,resM,t)}{cap(x,i')}.
\end{aligned}$$

If the tube demand between i' and e exceeds $cap(x,i')$, i.e.,

$$tubeDem(x,i',e,resM,t) \ge cap(x,i')$$

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. □

Per Request-Proportionality: Suppose two sources s and s' respectively make new reservations m and m' whose paths intersect on a connected segment $[v_0,\ldots,v_n]$, i.e.,

$$\begin{aligned}
&\exists n,k,k' \in \mathbb{N}.\\
&m.first < k \le m.last \ \wedge\ m'.first < k' \le m'.last\\
&\ \wedge\ \forall i \le n.\ v_i = m.path[k+i].as = m.path[k'+i].as
\end{aligned}$$

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

$$\begin{aligned}
&\forall v_i \in \{v_0,\ldots,v_n\}.\\
&\frac{ideal(m,resM_{v_i},\delta,t)}{ideal(m',resM_{v_i},\delta,t)} = \frac{ideal(m,resM_{v_0},\delta,t)}{ideal(m',resM_{v_0},\delta,t)}.
\end{aligned}$$

Lemma 13 (Per Request-Proportionality)**.**

$$\begin{aligned}
&\forall m,m' \in \mathcal{M}_R, r,r' \in Res, s,s' \in V, v \in H, i,e \in I, resM \in ResMap, t \in \mathbb{N}\\
&m.path[m.ptr] = m'.path[m'.ptr] = v_i^e \wedge\\
&s = src(m) \ \wedge\ s' = src(m') \wedge\\
&r = resM(v,s,m.id) \ \wedge\ r' = resM(v,s',m'.id) \wedge\\
&inDem(v,s,i,resM_v,t) \le \delta\cdot cap(v,i) \wedge\\
&egDem(v,s',e,resM_v,t) \le \delta\cdot cap(v,e)\\
&\Rightarrow\ \frac{ideal(m,resM,\delta,t)}{ideal(m',resM,\delta,t)} = \frac{idBW(r.vrs,t)}{idBW(r'.vrs,t)}
\end{aligned}$$

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

$$\begin{aligned}
&m.path[m.ptr] = m'.path[m'.ptr] = v_i^e\\
&s = src(m), s' = src(m')\\
&r = resM(v,s,m.id), r' = resM(v,s',m'.id)
\end{aligned}$$

and that s and s' have modest demands, i.e.,

$$(a)\ \ inDem(v,s,i,resM_v,t), inDem(v,s',i,resM_v,t) \le \delta\cdot cap(v,i)$$
$$(b)\ \ egDem(v,s,e,resM_v,t), egDem(v,s',e,resM_v,t) \le \delta\cdot cap(v,e).$$

By this it follows that both *inScalFctr* and *egScalFctr* are equal to 1 and therefore (c) $adjIdDem = idBW$ for m and m', respectively.

Using (a), (b), (c), and the definition of *ideal* it follows:

$$
\begin{aligned}
&ideal(m, resM, \delta, t)\\
&= reqRatio_{transit} \cdot linkRatio_{transit} \cdot tubeRatio \cdot \delta \cdot cap(v, e)\\
&= \frac{adjIdDem(v, r, resM, t)}{trDem}\\
&\cdot \frac{\min\{\delta \cdot cap(v, i), trDem\}}{\min\{\delta \cdot cap(v, i), stDem\} + \min\{\delta \cdot cap(v, i), trDem\}}\\
&\cdot \frac{\min\{\delta \cdot cap(v, i), tubeDem(v, i, e, resM, t)\}}{\sum_{i' \in I} \min\{\delta \cdot cap(v, i'), tubeDem(v, i', e, resM, t)\}} \cdot \delta \cdot cap(v, e)\\
&= \frac{idBW(r.vrs, t)}{stDem + trDem}\\
&\cdot \frac{tubeDem(v, i, e, resM, t)}{\sum_{i' \in I} \min\{\delta \cdot cap(v, i'), tubeDem(v, i', e, resM, t)\}} \cdot \delta \cdot cap(v, e)
\end{aligned}
$$

Due to the assumption $m.path[m.ptr] = m'.path[m'.ptr] = v_i^e$ it follows that *stDem*, *trDem*, and *tubeDem* are the same for m and m', and therefore the conclusion of this lemma. □

E.5 Global Properties of N-Tube

Given a valid execution, we formally specify the global properties (G1–G5). Table 1 shows a summary of these properties, which we define and prove in detail in the following.

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

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

Proof sketch. Follows directly from Theorem 1 and the *positivity* property of the bandwidth computation from Section 4.3. □

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

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

Proof sketch. This follows directly from the first statement in Theorem 1, since the *resBW* fields are set to $finBW(m)$ for all path ASes until m expires. □

Stability: During a period of constant demands, all reservations stabilize.

Corollary 3 (Stability)**.** Assume there are constant demands D between t_0 and t_1. Then

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

Proof sketch. This follows directly from Theorem 2. □

Minimum Bandwidth Guarantee: If there are constant demands 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 D between t_0 and t_1, then there is a lower bound $G * r$ on the ideal bandwidth allocation that all ASes on m's path reserve until m expires.

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

Proof sketch. This follows directly from Theorem 2 and the *lower* ideal *bound* property of the *ideal* function in the bandwidth computation from Section 4.3. □

Bounded Tube Fairness: If there are constant demands 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 D between t_0 and t_1, then

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

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

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

Here $tubeRatio_v$ and $tubeDem_v$ denote the corresponding functions defined in Section 4.3 that are computed at AS v.

Proof sketch. This follows directly from Theorem 2 and the *bounded tube-proportionality* property of the *ideal* function in the bandwidth computation from Section 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.

F PARAMETERS FOR STATISTICAL ANALYSIS

Table 2 lists all parameters used in the statistical analysis of N-Tube as described Section 6 together with their default values, which the three generators use to generate the topologies, paths, and workloads (including reservations, renewals, and deletions).

Table 2: Generators Parameters

Parameters	Default Value
# benign ASes	90
# malicious ASes	10
# sources	20
# intermediate ASes	75
# destinations	5
adjusted capacity δ	0.8
minBW	[0,50]
maxBW	[50,250]
maxT	20
reservation frequency	10
renewal frequency	2
deletion frequency	50
snapshot frequency	5
reservations per source	5
# paths per source-dest pair	5
segment length	5
link capacity	[100,300]
message delay	lognormal