Secure and Scalable QoS for Critical Applications

Marc Wyss, Giacomo Giuliari, Markus Legner, and Adrian Perrig
Department of Computer Science, ETH Zurich, Switzerland
{marc.wyss, giacomog, markus.legner, adrian.perrig}@inf.ethz.ch

***Abstract*—With the proliferation of online payment systems, the emergence of globally distributed consensus algorithms, and the increase of remotely managed critical IoT infrastructure, the need for *critical-yet-frugal* communication—high-availability and low-rate—is becoming increasingly pressing. For many of these applications, the use of leased lines or SD-WAN solutions is impractical due to their inflexibility and high costs, while standard Internet communication lacks the necessary reliability and attack resilience.**

To address this rising demand for strong quality-of-service (QoS) guarantees, we develop the GMA-based light-weight communication protocol (GLWP), building on a recent theoretical result, the GMA algorithm. GLWP is a capability-based protocol which is able to bootstrap network-wide bandwidth allocations in single round-trip times, and achieves high availability even under active attacks. Due to its clever use of cryptographic mechanisms, GLWP introduces minimal state in the network and causes low computation and communication overhead. We implement a GLWP prototype using Intel DPDK and show that it achieves line rate on a 40 Gbps link running on commodity hardware, thus showing that GLWP is a viable solution to provide strong QoS guarantees for critical-yet-frugal communications.

I. Introduction

With the steady advance of digitalization, the reliance on communication availability in critical sectors is increasing. This trend is further fueled by the move to digital payment systems, cloud infrastructure, and distributed IoT systems.

For many critical applications, reliability, security, and scalability of communication systems are of paramount importance. Yet, current approaches fall short of achieving all objectives simultaneously. Critical applications often rely on leased lines, sacrificing on scalability and flexibility while incurring high cost. On the other hand, best-effort Internet connectivity cannot satisfy the reliability and security requirements.

Many of these use cases can be characterized as *critical-yet-frugal* (CyF) applications: the traffic volumes exchanged are relatively low, but availability has to be guaranteed at all times for the service to achieve its mission. We provide two examples of such applications.

The first is inter-bank transactions. The SWIFT financial messaging network is of crucial importance for today's financial system, as it accounts for half of global cross-border inter-bank transactions [1]. However, the network's actual bandwidth requirements are modest: a recent estimate [2] shows that the average load on the network is likely less than 1 Mbps between all 11 000 of its member institutions.

The Bitcoin network provides a second example. Today, this network processes 7 transactions per second [3], with an average transaction size of 500 B, and very rarely above 1 kB [4]. This directly translates to modest bandwidth requirements of less than 100 kbps per node. In both these examples low-bitrate communications carry immense value and are therefore susceptible to attacks: it has been shown that disruptions can result in severe financial loss [5]. A further complication in securing communication in blockchain networks is decentralization: as nodes are run by many, often untrusted, entities, centralized solutions have to be avoided as they introduce a single point of failure. Even when centralized control could in principle be achieved, e.g., in permissioned blockchains, decentralization is used as a way to build trust [6].

Motivated by these examples, we define critical-yet-frugal applications to share these common traits: (i) the required traffic volumes are relatively small, less than 100 kbps per end-to-end communication, but (ii) connectivity has to be ensured at all times (availability), (iii) even in the presence of denial-of-service (DoS) attacks (security). Finally, (iv) the guarantees have to be extended to large networks, even under the assumption of decentralized control. These peculiar characteristics make existing solutions ill-suited to support CyF traffic.

Why are new ideas needed? Two main avenues exist to satisfy CyF applications: either deploying—or leasing—dedicated communications infrastructure (i.e., completely isolating traffic), or implementing a protocol-based quality-of-service (QoS) system over existing infrastructure. Unfortunately, both directions have shortcomings:

- *High cost:* Dedicated infrastructure provides the highest security guarantees due to complete traffic separation. However, this comes at high cost, translating to low redundancy and thus reduced availability.
- *Central control:* Most forms of QoS, as well as dedicated infrastructure, require centralized orchestration. This is not a drawback per-se, but can clash with the requirements of decentralized applications.
- *No protection against adversaries:* Most protocol-based QoS solutions are designed under the assumption of a benign environment, in which the only threat is congestion.
- *Network state:* Even in non-adversarial environments, QoS mechanism such as *IntServ* heavily rely on in-network state. Nodes have to keep state for all the allocations they provide—usually at the flow level—which causes inherent scalability issues. In fact, in-network state has been identified as a threat to the deployment of network protocols [7].

Unconditional resource allocation. In this paper, we side-

978-0-7381-3207-5/21/$31.00 ©2021 IEEE

step these obstacles by leveraging the *unconditional resource allocation* of the global myopic allocation (GMA) algorithm [2]. GMA allows to compute non-zero bandwidth allocations for *any* allowed path in a network. These allocations are unconditional, i.e., the amount of allocated bandwidth for one path is independent of the allocated bandwidth on all the other paths. At the same time, GMA guarantees that—even if all allocations in the network are fully utilized—no link will ever be congested. To calculate the allocation of a specific path, GMA purely relies on information of on-path nodes, and therefore no global coordination is needed. Although an allocation is diminishing with increasing path length, these allocations are sufficient to carry and protect CyF traffic in real networks. Moreover, best-effort traffic can make use the of the (potentially) unused GMA bandwidth, as we describe in §V, further increasing network utilization and efficiency.

We concretize this theoretical result and propose a secure, efficient bandwidth-allocation protocol for the Internet: the GMA-based light-weight communication protocol (GLWP). Since GMA only requires path-local information, GLWP can collect this information and compute the corresponding allocation efficiently in a *discovery phase* in the control plane. These allocations are then disseminated to traffic sources in the form of cryptographic tokens, eliminating the need for per-allocation state in the network. As GMA allocations guarantee congestion-freeness, monitoring and policing traffic in the data plane is sufficient to enforce correct use of the allocations. GLWP is very efficient and highly scalable, and able to saturate today's high-speed (40 Gbps) network links even on commodity hardware.

This paper makes the following core contributions:

- We present GLWP, the first distributed protocol to achieve forwarding guarantees for any path in the Internet, even in adversarial settings.
- We modify the original GMA algorithm and apply it to the Internet, introducing realistic assumptions on the operation of autonomous systems (ASes).[1] Most importantly, we extend the algorithm with the notion of time.
- We analyze the security and availability properties of GLWP, and how they arise from GMA allocations.
- We implement and evaluate GLWP, demonstrating scalability and high throughput on commodity hardware.

II. Background: unconditional allocations

In this section, we summarize the main ideas of the GMA algorithm, and the details necessary to understand GLWP's operation. The GMA paper [2] contains the full details.

A. The GMA algorithm

Allocation graphs and computer networks. The GMA algorithm is defined on *allocation graphs*, which are composed of nodes and edges. Each edge has assigned a non-negative amount called *resource*. In this paper, we are interested in the applications of GMA to digital communications and QoS; therefore, we identify the graph with the Internet, nodes with ASes, and the resource with link bandwidth. Every node is augmented with a set of *interfaces*, each of which represents the interconnection point between two nodes; concretely, in our setting, interfaces are connections among ASes at border routers. An additional *local interface* represents the traffic sources and destinations internal to the node.

A *path* π in an allocation graph is specified to the granularity of interfaces, i.e., it is the sequence of interfaces traversed by the path. For any node on the path, the *interface pair* denotes the interfaces where the path enters and exits the node. In an allocation graph, each node further defines an *allocation matrix* (see §II-B) which captures the local decision, or policy, on the maximum bandwidth each neighbor is allowed to allocate on the node's links.

Algorithmic objective. The goal of GMA is to compute the size of a bandwidth allocation for *any* specific path on the network, reserving it for the communication between the source and destination nodes of the path. Most importantly, allocations are unique to a specific path and are not granted, for example, to single flows. Therefore, two communicating nodes can use a single allocation per path. These allocations resemble a tunnel between source and destination nodes, with two important properties: the allocation 'tunnel' is tied to the path it was granted for, and the bandwidth granted to this tunnel is fixed. This greatly reduces complexity, and avoids the pitfalls that arise when implementing per-flow allocations [8].

GMA's properties. The GMA algorithm provides an essential property to our goal of supporting network-wide QoS for CyF applications: it ensures that the allocations are sized in such a way that, even if *all* paths in the network receive an allocation, and these are fully utilized, *no link is ever congested.* This property, called *no-over-allocation* in the GMA paper, is mathematically proven to hold for all networks. Further, allocations are computed based only on *information local to the path*, i.e., without the need for global coordination.

These strong guarantees are achieved at the expense of flexibility: the size of the allocation is determined by the algorithm, run on all on-path nodes, and cannot therefore be adapted to communication requirements. Interestingly, GMA allocations can be thought of as an *emergent behavior*: they arise from the purely local, myopic policies of individual nodes and exist implicitly at all times irrespective of whether they are used, until they are finally "discovered" in the control plane.

It may seem that the inflexible and relatively small amount of bandwidth computed by GMA might pose an obstacle in practice. However, the GMA paper shows that in real-world networks these allocations suffice to support CyF traffic.

B. Essential details of GMA

Allocation matrices. Each node k defines an allocation matrix, $M^{(k)}$, in which every entry $M^{(k)}_{i,j}$ represents the *maximum amount of bandwidth* that the node is willing to allocate to all the paths transiting from interface i to interface j. Given

[1] ASes are the centrally-operated networks that interconnect through an exterior gateway protocol and form the Internet.

this matrix, the maximum amount of bandwidth that can be allocated from an interface i to any other interface within the node is called *divergent*, and the maximum amount of bandwidth that can be allocated from any interface in the node towards interface j is called *convergent*:

$$DIV_i^{(k)} = \sum_j M_{i,j}^{(k)}, \qquad CON_j^{(k)} = \sum_i M_{i,j}^{(k)}. \quad (1)$$

Allocation computation. Starting from allocation matrices, the GMA allocation $\mathcal{G}(\pi)$ for some path π (expressed as a sequence of ℓ interface pairs) can be computed using Eq. (10) in the GMA paper:

$$\mathcal{G}(\pi) = \min_{1 \le x \le \ell} \left(\prod_{k=1}^{x-1} \frac{M_{i,j}^{(k)}}{CON_j^{(k)}} \cdot M_{i,j}^{(x)} \cdot \prod_{k=x+1}^{\ell} \frac{M_{i,j}^{(k)}}{DIV_i^{(k)}} \right) \quad (2)$$

Specifically, three values are needed as input from each on-path node $k \in \{1, \ldots, \ell\}$: $M_{i,j}^{(k)}$, $DIV_i^{(k)}$, $CON_j^{(k)}$, given that the path traverses node k, incoming at interface i and outgoing at interface j. These three values are also referred to as the *hop values*. Finally, GMA also allows to derive an upper bound on the allocation $\mathcal{G}(\pi)$, called *preliminary guarantee*, by computing an allocation for any k-subprefix of the path, i.e., using the hop values of the first k nodes only. This is referred to as the *extensibility property*.

The GMA paper does not specify, however, how allocations can be created and used in practice. Many questions arise when considering concrete applications of the algorithm: How can nodes exchange hop values? How can allocation matrices be updated to reflect changes in the network, without causing over-use? How can we secure the allocation computation and the data traffic from adversarial action? How can this be done without causing excessive computation and communication overhead? GLWP is a solution to these open problems.

III. Requirements and architecture

Although many different types of networks can in principle support GMA-based allocations, our aim is to design a protocol that enables CyF communications in the Internet. Therefore, we instantiate nodes with ASes, which are seeking to provide a highly-available communication service alongside best-effort traffic. Edges then map to inter-domain links, and interfaces to border routers.

In the following, we specify the security and efficiency requirements for GLWP, as well as the adversary and network models we consider for our design. From now on, we will only refer to non-source ASes as *on-path ASes*.

A. Adversary model

We consider a Dolev–Yao adversary [9] that can intercept, modify, drop, and inject packets anywhere in the network but cannot break cryptographic primitives. We further assume that some of the ASes are malicious, but that those ASes still protect themselves from other ASes, meaning that they actively monitor misbehavior (announcements of inconsistent hop values). The *adversary's objective* is to interfere with the protocol such that the intended properties are violated.

B. Network model

AS structure. To participate in GLWP, ASes must compute their allocation matrices based on local policies and historical data—e.g., the traffic volumes from and to neighboring ASes. Internet service providers (ISPs) carry out a similar procedure already today, as they need to build traffic matrices to provision internal routing [10]. The operations required by the GLWP control plane are (logically) centralized in a *GLWP service* (GServ) in each AS, while, in the data plane, border routers have to support GLWP traffic.

Path stability. A crucial prerequisite for any system that provides meaningful allocations for specific paths is *path stability*: if forwarding paths change frequently, or can be easily disrupted by external action, no guarantee can be provided for traffic on those paths. In centrally managed networks, path stability can be achieved through, for example, software-defined networking (SDN) [11] or MPLS [12]. Segment routing [13] can provide path stability even for larger networks. In the context of inter-domain routing, where central coordination is not available, the SCION Internet architecture provides AS-level path stability [14].

Key management. In order to fulfill the protocol requirements in adversarial settings, a layer of (cryptographic) measures needs to be implemented. Unfortunately, asymmetric cryptography is computationally expensive and enables additional attack vectors such as signature-flooding attacks. We therefore rely on systems that efficiently distribute symmetric keys between any pair of ASes such as Passport [15] or PISKES [16] in an inter-domain context, or Kerberos [17] if a trusted authority exists. Irrespective of the key-distribution mechanism, storage requirements are modest: for example, storing a 16 B symmetric key for 10 000 ASes only requires 160 kB. We denote the shared symmetric keys between two ASes k and m as $K_k^m (= K_m^k)$. Since GMA allocations depend on the allocation-matrix entries of all on-path ASes, we use message-authentication codes (MACs) based on the pairwise shared symmetric keys to authenticate this information (§V-A). Furthermore, we assume that every AS k has a secret key K_k, which is known to AS k alone. Finally, we assume that ASes are time-synchronized with second-level precision and that an in-network duplicate-suppression system scrubs replayed packets [18].

C. Protocol requirements

Function and security. For arbitrary paths (every AS might potentially be malicious) we require the following properties:

- **F1** ASes only communicate the information necessary to compute the bandwidth allocation. This is to prevent excessive disclosure of sensitive information.
- **F2** ASes cannot communicate inconsistent values that could cause an overuse of capacities anywhere in the network.

For paths with *only benign on-path ASes* (the source may be malicious) we additionally require the following:

F3 A source AS can only obtain one guarantee per path.
F4 The source cannot overuse the bandwidth guarantee.

For a path consisting *only of benign ASes*, GLWP must further satisfy the following requirements:

F5 The bandwidth guarantee calculated by the ASes is given by the GMA algorithm.
F6 Each guarantee is only valid for one specific path.
F7 Malicious off-path ASes cannot disrupt the guarantee.
F8 No AS can frame the source of overusing its guarantee.

Finally, for a path with *only benign ASes and no attacker on the links*, GLWP must achieve the following properties:

F9 No off-path attacker can prevent the source AS from obtaining a guarantee.
F10 The delivery of data traffic is guaranteed.

The restriction to benign entities is necessary as some could drop packets to trivially interrupt communication.

Efficiency. There are several non-functional requirements:

E1 Data-plane packets must be processed within hundreds of nanoseconds to achieve line rate on gigabit links.
E2 Routers in the dataplane only need to store a constant amount of state, irrespective of the number of ASes, the number of flows, or the number of paths.

Summing up. Assuming a network that provides path stability and an adequate key-management system, the role of GLWP is to (i) gather and authenticate the necessary information to compute the allocation with GMA, and (ii) protect forwarding. We will use the convention that a path consists of a *source AS* and multiple *on-path ASes*. As the length of a path is ℓ, there are $\lambda := \ell - 1$ on-path ASes. To calculate an allocation using GMA, each AS only needs to contribute three terms: the divergent of the incoming interface, the allocation-matrix entry for the interface pair, and the convergent of the outgoing interface—i.e., the hop values $(DIV_i,\ M_{i,j},\ CON_j)$.

IV. GLWP overview

GLWP is designed starting from two key insights:

1) GMA allocations are computed only on *path-local* hop values, which are then the only values that the allocation protocol has to distribute and authenticate. This removes the need for global coordination.
2) Since GMA allocations are unconditional and can never cause over-allocation, each allocation can be granted independently of already existing allocations without the risk of causing congestion. Thus, each AS does not need to keep track of the amount of bandwidth it allocates. This greatly reduces the computation overhead and state required at GServs, and enables stateless border routers.

We now present how these insights are used in GLWP.

A. Control- and data-plane overview

In the *discovery phase* (Fig. 1a), the computation and authentication of the allocations is carried out in the control plane. The source initiates the process by sending a discovery packet towards the destination. The GServs in on-path ASes append their hop values for the computation of the allocation, and forward the modified and authenticated packet along the path (insight 1). Once the destination receives the packet, it is sent back along the same path, so that all ASes can gather the authenticated hop values, compute the allocation *using GMA*, and append capabilities—authorizations to use the allocation—in the form of cryptographic keys for the source. It is important to note that the on-path ASes can *forget the allocations and the capability tokens they have granted*, and do not need to store them.

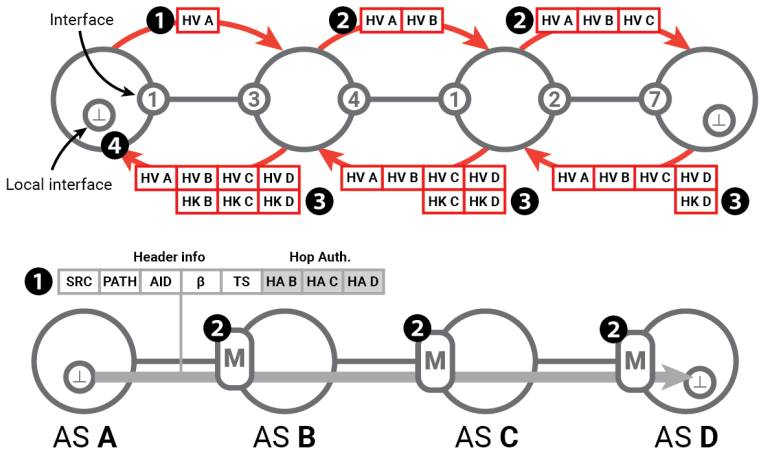
In the *transmission phase* in the data plane (Fig. 1b), the source encapsulates the application payload in GLWP packets, which are then further encapsulated in network-layer packets. Using the previously received cryptographic keys, the source calculates authenticators for every AS on the path. The border routers of those ASes can subsequently check the validity of the allocation and the authenticity of the source and the GLWP header statelessly on the fly by recomputing the authenticators (insight 2). As the GLWP header contains the size of the allocation that was computed in the control plane, each on-path AS can monitor and enforce the allocations.

B. Enforcing QoS

CyF applications are often deployed in a decentralized, adversarial setting. Our system is therefore designed to provide strong guarantees even in the presence of adversaries. Two attack vectors are open for an adversary to disrupt GLWP communications: tampering with the control-plane setup phase in order to, e.g., obtain larger allocations, and volumetric DoS attacks with GLWP traffic. The first attack is prevented by mutually authenticating all the values and entities participating in the allocation setup. The second attack, DoS with GLWP traffic, is precluded because (i) GMA computes allocations that will never cause over-allocation at any link (where the correctness of those calculations is enforced in the control plane), and (ii) the GLWP traffic is monitored (which is possible as every single packet is authenticated) to check that allocations are not abused. We implement monitoring with two systems found in the literature: highly scalable probabilistic bandwidth monitor [19], complemented by a replay-suppression system that avoids framing attacks with spoofed packets [18].

Thanks to recent advances in key-distribution protocols [16], the cryptographic operations can be performed efficiently even at line rate. We describe further details in Sections V-A and V-B and evaluate a prototype implementation in §VII. The adversary model and security landscape is defined in §III-A.

The role of GLWP is therefore to *authentically distribute* the hop values of each AS on a path to all others, such that they can *locally use the GMA algorithm to compute the allocation size*. GLWP then creates cryptographic authorizations (capabilities) for the source to use this allocation, and enforces them in the data plane. With source authentication and monitoring, GLWP further ensures that the allocated bandwidth is not exceeded.



(a) *Discovery phase:* ❶ *AS A initiates the discovery on the path (π), sending its Hop Values (HV);* ❷ *ASes append HVs to the packet, authenticated to all other ASes on the path;* ❸ *on the return path, ASes use HVs to compute the allocation $\mathcal{G}(\pi)$ and authenticate it by creating Hop Keys (HK); HKs are encrypted and appended to the packet; on-path ASes do not need to store HVs nor HKs;* ❹ *AS A stores the HKs for use in the transmission phase.*
(b) *Transmission phase:* *AS A wants to send a packet to AS D;* ❶ *it generates Hop Authenticators (HA) starting from the HKs and the packet header, and appends them to the packet;* ❷ *each AS verifies that the header of the packet matches its HA, and uses a traffic monitor and a duplicate-suppression system to ensure that the communication does not exceed the allocation $\mathcal{G}(\pi)$.*

Figure 1: ***Overview of GLWP, discovery phase and transmission phase.***

V. Protocol Description

We now proceed to present the detail of GLWP. In the *discovery phase* (control plane, Algorithm 1), an AS discovers the bandwidth guarantee on a chosen path and obtains credentials for using it to send data traffic during the subsequent *transmission phase* (data plane, Algorithm 2). Finally, we provide a sub-protocol to handle updates to allocation matrices.

A. Discovery phase

As every AS on the path should be able to authenticate the data for every other AS, λ^2 MACs need to be calculated and transmitted for a path consisting of ℓ ASes. To avoid excessive communication overhead, we propose to use a single authentication field per AS (so ℓ fields in total) together with aggregate MACs [20], [21]. All the following operations are carried out by the GServ of the respective ASes.

At the beginning of the discovery phase, the source AS selects a path π (interface pairs) and a sequence of corresponding node identifiers (AS numbers, NId for short), denoting the ASes that it intends to use. It then deterministically constructs an *allocation identifier* (AID) by hashing the path, so $AID = \mathsf{H}(\pi)$. Using the shared symmetric keys, the source authenticates its information, which includes its hop values, by calculating an authenticator for each on-path AS.

Upon receiving the packet, an on-path AS recalculates and verifies the AID. The AS validates the authenticity of all the previous hop values using the aggregate MACs, appends its own hop values to the packet, and adds authenticators for every other AS to the aggregate MACs. Using GMA, it calculates a preliminary bandwidth guarantee (see §II-B) based on the hop values discovered so far. If it is smaller than some threshold, i.e., it is too low to be usable, the AS sends an error message to the source, otherwise it forwards the packet to the next on-path AS. The last AS on the path sends the packet back towards the source, on the reversed path.

On the backward path each AS again checks whether all the information is authentic and that the announced allocation-matrix entries do not exceed the corresponding convergent and divergent. Also, every AS stores the announced hop values of each of its neighbors. If those are inconsistent across time, the AS can take action against this misbehavior. For example, it can refuse any further allocation requests coming through that neighbor. Announced hop values are *inconsistent*, if one of the following situation arises: (i) for some interface pair of the neighbor different hop values were announced, (ii) for the neighbor's interface that connects it to the AS executing the consistency checks, different divergents/convergents or interface identifiers were announced, or (iii) the sum of the announced matrix entries for some interface exceeds the divergent/convergent corresponding to that interface.

As long as the hop values are consistent, there will never be an over-allocation of any link (this follows from GMA's no-over-allocation property). Each on-path AS then uses the hop values to compute the final bandwidth allocation according to GMA. It also derives a *hop key* (HK) for the source, encrypts and authenticates it, and adds it to the packet. The hop keys created by the on-path ASes represent *capabilities* that allow the source to send traffic along this path using the guaranteed bandwidth, until the hop keys expire.

When the source finally receives the packet, it checks the authenticity of the information, and decrypts the hop keys. It stores the AID, the hop keys, the expiration time, and the allocated bandwidth together with the path.

B. Transmission phase

During the transmission phase, the source AS uses the hop keys it received previously in order to calculate a per-packet *hop authenticator* (HA) for each on-path AS. The hop authenticators are different for each packet, because they depend on the source AS identifier (Src) and a packet timestamp (ts_{pkt}), which together uniquely define a packet. The hop authenticators then allow on-path ASes to authenticate the source and verify the length of the packet. If its dedicated hop authenticator is valid, each on-path AS passes the Src, AID, bandwidth allocation $\mathcal{G}(\pi)$, as well as the packet length to a separate probabilistic traffic monitor [19] and forwards the packet to the next AS on the path. The traffic monitor

Algorithm 1 Discovery phase. Packet fields are denoted by $\boxed{\text{FIELD}}$ and $\leftarrow$ is an assignment. To ensure recentness of packets, the on-path GServs compare the timestamp to the current time while allowing a lifetime L (hundreds of milliseconds), taking into account a clock skew ϵ (up to a few seconds). τ denotes the minimum useful guarantee value. L, ϵ, and τ are globally fixed parameters.

1: **procedure** PACKET INITIALIZATION AT THE GSERV OF THE SOURCE AS
Require: Src (= NId_1), NId_2, …, NId_ℓ, π (path), HV_1, K_1^2, …, K_1^ℓ
2: $ts_{\text{pkt}} \leftarrow$ (current time)
3: $AID \leftarrow \mathsf{H}(\pi)$
4: $Auth_1 \leftarrow 0$
5: **for** $m \in \{2, \ldots, \ell\}$ **do**
6: $Auth_m \leftarrow \mathsf{MAC}_{K_1^m}(Src,\ ts_{\text{pkt}},\ AID, HV_1)$
7: Create packet with Src, $NId_2, \ldots, NId_\ell$, ts_{pkt}, AID, $Auth_1, \ldots, Auth_\ell$, and HV_1

8: **procedure** APPENDING HOP VALUES BY ON-PATH AS k (FORWARD PATH)
Require: HV_k, $\forall m \in \{1, \ldots, \ell\} \setminus \{k\} : K_k^m$
9: **if** packet was not received through interface $\boxed{i_k}$ **then** drop packet
10: **if** $\boxed{NId_{k-1}}$ or $\boxed{NId_{k+1}}$ do not match AS k's neighbors **then** drop packet
11: **if** (current time) $-$ $\boxed{ts_{\text{pkt}}} \notin [-\epsilon, L+\epsilon]$ **then** drop packet
▷ invalid timestamp (lifetime L, clock skew ϵ)
12: $AID \leftarrow \mathsf{H}\left(\boxed{\pi}\right)$
13: **if** $AID \neq \boxed{AID}$ **then** drop packet ▷ invalid allocation ID
14: $Auth_k \leftarrow \bigoplus_{m \in \{1,\ldots,k-1\}} \mathsf{MAC}_{K_m^k}(Src,\ ts_{\text{pkt}},\ AID, HV_m)$
15: **if** $Auth_k \neq \boxed{Auth_k}$ **then** drop packet
16: $\boxed{HV_k} \leftarrow HV_k$
17: **for** $m \in \{1, \ldots, \ell\} \setminus \{k\}$ **do**
18: $Auth_m \leftarrow Auth_m \oplus \mathsf{MAC}_{K_m^k}(Src,\ ts_{\text{pkt}},\ AID, HV_k)$
19: calculate preliminary guarantee using $\boxed{HV_1}, \ldots, \boxed{HV_k}$ ▷ §II-B
20: **if** preliminary bandwidth $< \tau$ **then**
21: send error packet back to source AS
22: **else**
23: forward packet to the next AS according to $\boxed{\pi}$

24: **procedure** HOP KEY GENERATION BY ON-PATH AS k (RETURN PATH)
Require: HV_k, K_k, $\forall m \in \{1, \ldots, \ell\} \setminus \{k\} : K_k^m$
25: repeat checks from lines 9-13
26: **if** $HV_k \neq \boxed{HV_k}$ **then** drop packet
27: $ts_{\text{exp}} \leftarrow \boxed{ts_{\text{pkt}}} + \Delta T$
28: $Auth_k \leftarrow \bigoplus_{m \in \{1,\ldots,\ell\} \setminus \{k\}} \mathsf{MAC}_{K_k^m}(Src,\ ts_{\text{pkt}},\ AID, HV_m)$
29: **if** $Auth_k \neq \boxed{Auth_k}$ **then** drop packet
30: **for** $m \in \{1, \ldots, \ell\}$ **do**
31: **if** $\left(M_{im,jm}^{(m)} > DIV_{im}^{(m)}\right) \vee \left(M_{im,jm}^{(m)} > CON_{jm}^{(m)}\right)$ **then**
32: drop packet / blame AS m
33: check that the hop values promoted by the neighboring ASes are consistent
34: calculate final bandwidth guarantee $\beta \leftarrow \mathcal{G}(\pi)$ ▷ §II-B
35: $HK_k \leftarrow \mathsf{MAC}_{K_k}(Src, AID, \beta, i_k, j_k, ts_{\text{exp}})$
36: $IV \leftarrow (Src, AID, ts_{\text{pkt}})$
37: $(Ciph_k, Tag_k) \leftarrow \mathsf{AuthEnc}_{K_k^1}(IV, HK_k)$
38: $\boxed{Ciph_k} \leftarrow Ciph_k$
39: $\boxed{Auth_1} \leftarrow \boxed{Auth_1} \oplus Tag_k$
40: forward packet to the next AS according to $\boxed{\pi}$

41: **procedure** STORING HOP KEYS AT THE GSERV OF THE SOURCE AS
Require: K_1^2, …, K_1^ℓ
42: repeat checks and calculations from lines 25-27 (with $k = 1$)
43: **for** $m \in \{2, \ldots, \ell\}$ **do**
44: $(HK_m, Tag_m) \leftarrow \mathsf{AuthDec}_{K_1^m}(\boxed{IV}, \boxed{Ciph_m})$
45: $Auth_1 \leftarrow \bigoplus_{m \in \{2,\ldots,\ell\}} \left(\mathsf{MAC}_{K_1^m}(Src,\ ts_{\text{pkt}},\ AID, HV_m) \oplus Tag_m\right)$
46: repeat checks from lines 29-34 (with $k = 1$)
47: store the AID, the hop keys, β, and ts_{exp} as attributes of the path

Algorithm 2 Transmission phase. Syntax as in Algorithm 1.

1: **procedure** PACKET GENERATION AT THE GSERV OF THE SOURCE AS
Require: Src, π (path), $HK_1, \ldots, HK_\ell$, AID, β, ts_{exp}, P (payload)
2: $ts_{\text{pkt}} \leftarrow$ (current time)
3: pkt$\leftarrow$ (create new packet with reserved space for header and payload)
4: *pkt-len* $\leftarrow \text{length}(\text{pkt})$
5: **for all** $k \in \{2, \ldots, \ell\}$ **do**
6: $HA_k \leftarrow \mathsf{MAC}_{HK_k}(Src, ts_{\text{pkt}}, \textit{pkt-len})[\![0{:}l_{\text{HA}}]\!]$
7: create packet with Src, π, AID, β, ts_{pkt}, ts_{exp}, $HA_2, \ldots, HA_\ell$, and P
8: send packet to the next AS according to π

9: **procedure** VALIDATION AT BORDER ROUTER OF ON-PATH AS k
Require: K_k
10: **if** packet was not received through interface $\boxed{i_k}$ **then** drop packet
11: **if** (current time) $-$ $\boxed{ts_{\text{pkt}}} \notin [-\epsilon, L+\epsilon]$ **then** drop packet
12: **if** $\boxed{ts_{\text{exp}}}$ < (current time) + L+ ϵ **then** drop packet ▷ allocation expired
13: $HK_k \leftarrow \mathsf{MAC}_{K_k}(Src, AID, \beta, i_k, j_k, ts_{\text{exp}})$
14: $HA_k \leftarrow \mathsf{MAC}_{HK_k}(Src, ts_{\text{pkt}}, \textit{pkt-len})[\![0{:}l_{\text{HA}}]\!]$
15: **if** $\boxed{HA_k} \neq HA_k$ **then** drop packet
16: pass (Src, AID, β, *pkt-len*) to traffic monitor
17: pass (Src, ts_{pkt}) to duplicate-suppression system
18: forward the packet according to $\boxed{\pi}$

detects when the source overuses the bandwidth guarantee for the path identified by the AID. In such a case, the AS can take measures such as blacklisting the source.

C. Expiration and traffic-matrix updates

To enable topology changes and modifications to allocation matrices, GLWP guarantees expire after a fixed amount of time ΔT. As the topology and allocation matrices are expected to be relatively stable and to minimize overhead due to discovery packets, we propose to set ΔT to one day. If an AS wants to modify its allocation matrix at some time t_1, there may be valid "old" GLWP guarantees until $t_2 = t_1 + \Delta T$; only at t_2 the definitive change in values may occur. To prevent over-allocation, the AS needs to announce temporary hop values during $[t_1, t_2]$, which consist of the maximum of the old and new divergent and convergent and the minimum of the old and new allocation-matrix entry. This ensures that the temporary guarantees calculated with GMA are smaller than or equal to guarantees calculated based on the old and new allocation matrix (see Eq. (2)); in particular, over-allocation cannot occur in the intervals $[t_1, t_2]$ (where *old* and *temporary* allocations exist simultaneously) and $[t_2, t_2 + \Delta T]$ (where *temporary* and *new* allocations exist simultaneously).

Because such changes in the announcements would contradict the consistency checks described in §V-A, we need to adapt those checks to allow changes in the allocation matrix. For that, the ASes implement the following protocol:

Consistency-check subprotocol.

1) Every time an AS wants to update its allocation matrix (also when it joins the network for the first time), it authentically sends the row and column corresponding to a particular neighbor towards that neighbor. For a neighbor connected to interface i, this would be the ith column and the ith row of the allocation matrix.
2) The neighbor checks the authenticity of this update message and stores row and column, as well as their

sum (convergent and divergent), with a current timestamp. In order to limit space and prevent resource exhaustion attacks, it rate-limits the number of updates.
3) For each setup request in the discovery phase, the AS promotes the modified hop values as described in §V-C.
4) When a neighbor receives such a setup request, it fetches the convergent, divergent, and matrix values stored in step 2; then, it compares them to the hop values that were promoted in the request (step 3).

Note that this row and column of a matrix is exactly the information that the corresponding neighbor might learn anyway through different setup requests, and thus the subprotocol does not weaken property F1.

D. Notes on monitoring

The purpose of the traffic monitor is to enforce bandwidth allocations [19]. The monitor was designed such that it can support a wide range of protocols—in the case of GLWP, we can directly deploy the monitor by setting the *flow ID key* to the AID field (hash over the path) of the GLWP packets. The required state is minimal and fits into L2 or even L1 cache, and the monitor can handle traffic in the order of tens of Gbps. Higher throughput can be achieved by distributing flows among multiple copies of the system.

The in-network duplicate-suppression system, which uses Bloom filters to identify duplicates, can filter out replayed packets at traffic rates of 10 Gbps using only two cores on a commodity machine [18]. The packet timestamp (ts_{pkt}) of GLWP can serve as the required *packet sequence number*. Thanks to timestamps, the system only needs to keep track of recent packets (and discard old packets), which makes this process practical even for high-bandwidth applications.

Both the traffic monitor and the duplicate-suppression system assume network-layer source authentication, which GLWP provides by (i) authenticating the source in the discovery phase and (ii) sending the hop keys only in encrypted form to the source, thus establishing another shared secret. These systems and GLWP can but do not necessarily have to be executed on the same machine.

VI. Availability and scalability analysis

We now show that GLWP can withstand adversarial action both in the control and data plane and after that analyze the scalability of the discovery phase. A comprehensive evaluation of the transmission phase can be found in §VII.

A. Discovery-phase attacks

On-path network link attacks. During the discovery phase, an adversary could tamper with the hop values, attempting to influence the allocation size resulting from GMA. The mutual authentication of hop values prevents this attack (F5).

Malicious parameter announcement. Malicious ASes could report contradicting hop values to other ASes in order to induce congestion by violating the assumptions for GMA's no-over-allocation property. We protect against this attack by having ASes monitor the hop values of their neighbors over time. Inconsistent announcements can then be punished by refusing GLWP allocation requests from/to the neighbor. Therefore, property F2 holds.

Multiple allocation requests. A source AS may try to get multiple allocations for the same path by issuing multiple setup requests. However, even though the source will receive different hop keys for the same path (the hop key is calculated based on a fresh timestamp: line 35 in Algorithm 1), the calculated bandwidth and the AID will still be the same. The bandwidth monitor will therefore still account the packets authenticated with different hop keys to the same allocation, so a source cannot send more traffic on the same path (F3).

Path manipulation. An attacker could try to manipulate the path field (list of interface pairs) in the discovery phase packets in order to get an allocation based on a different AID. If such a manipulation is performed by an attacker on a link or some malicious on-path AS, then this will be detected by other ASes, because the path field is authenticated from every AS towards every other AS on the path. If such a manipulation is performed by a malicious source however, the modification can affect the interface pair of the source or the interface pair of some on-path AS. The first case can only have negative implications (potential over-allocation) for the other ASes if the source promotes inconsistent values, which is checked by the first on-path AS. In the second case, the corresponding on-path AS will either receive the setup request on a wrong interface and drop the packet, or forward it to a wrong interface (effectively changing the path for the allocation). In any case it is not possible to violate property F3.

Measures against denial of capability (DoC). While GLWP data-plane traffic cannot be affected by DoS attacks, the discovery packets need to be sent as best-effort traffic unless previous hop keys are available. To rule out DoC attacks, we propose to leverage GMA and its allocation matrices: Each AS can implement rate limiting for GLWP requests, where each of its neighbors i is allowed a rate of requests proportional to DIV_i. By this, the properties of GMA are transferred to the discovery phase and no AS can prevent other ASes from obtaining GLWP guarantees (F9).

B. Transmission phase attacks

In the following, we cover the attack surface of the transmission phase and show how we defend against each attack.

Overuse. A malicious source trying to overuse the bandwidth will be detected by the bandwidth monitor (F4).

Framing attack with spoofed packets. An attacker may try to spoof the packet source and to subsequently cause bandwidth overuse. Since the overuse of an allocation is detected by the monitoring subsystem and leads to allocation revocation, this attack leads to the DoS of a benign AS. This event is prevented by per-packet source authentication.

Framing attack with replayed packets. Similarly to the previous attack, the adversary tries to blame a benign AS for bandwidth overuse. In this case, however, they capture and replay valid packets from the legitimate source, thus correctly passing the checks for source authentication. However, the in-network duplicate-suppression system prohibits this option, and therefore precludes framing benign source ASes (F8).

Volumetric (D)DoS with best-effort and GLWP traffic. When trying to prevent the legitimate use of a GLWP allocation for specific path, off-path ASes can send traffic along (i) an unrelated path (a path with different interface pairs); (ii) along a path that has at least one common interface pair using best-effort traffic; or (iii) along a path that has at least one common interface pair using a different GLWP allocation.

Case (i) trivially does not present a threat for the communication. There may still be an intradomain overlap. However, this has to be taken into account when calculating the allocation matrix. In case (ii), QoS is maintained by using different queues for GLWP and best-effort traffic, effectively isolating the legitimate communication from the attack traffic. Finally, case (iii) is protected by GMA's *no-over-allocation* property: since both the adversary's and the benign allocation are legitimate—i.e., they have been computed in the authenticated discovery phase with legitimate values—they cannot create mutual congestion. Monitoring will detect and block an adversary trying to abuse such allocation. Therefore, malicious off-path ASes can not disrupt the bandwidth guarantees (F7).

Hop-key reuse. Using the hop keys issued for some path also for another path is not possible, because at least one on-path AS will rederive a different hop key based on its own, different private key (line 13 in Algorithm 2), which results in an authentication failure. Thus, property F6 is satisfied.

C. Scalability

During the discovery and transmission phase, on-path ASes only need to store their allocation matrix, the mutually shared symmetric keys, their secret key, and the promoted hop values (to check consistency). Even for Internet-scale networks, the required state is only in the order of megabytes (E2).

Discovery-phase scalability. For this phase, processing speed is of secondary importance, as only a single discovery packet is sent per connection and it is handled by the GServ, outside the routers' fast path. We therefore provide a complexity analysis and leave the implementation to future work. During the discovery phase, every AS authenticates its hop values to every other AS, and conversely checks the authenticity of the hop values of every other AS. In total, the discovery phase requires $\mathcal{O}(\ell^2)$ flops, $\mathcal{O}(\ell^2)$ MAC computations, $\mathcal{O}(\ell)$ encryptions and decryptions, as well as $\mathcal{O}(\ell)$ other checks and memory lookups. This complexity is equivalent to previously proposed data-plane protocols like ICING [21]. The current average path length in the Internet is 4–5 AS-level hops [22], therefore resulting in low overhead. The discovery phase only requires one round-trip, after which the source can start sending GLWP packets using its guaranteed bandwidth.

Table I: Size (in bytes) and number of occurrences (#) of the data-plane GLWP header fields for $\lambda = \ell - 1$ on-path ASes.

field	content	#	size
Src	source AS identifier (=NId_1)	1	4
λ	number of on-path ASes ($\lambda = \ell - 1$)	1	1
HP	hop pointer: number of already traversed ASes	1	1
π	path: array of interface pairs (2 B / interface)	ℓ	4
β	guaranteed bandwidth	1	4
AID	allocation identifier	1	8
ts_{pkt}	packet timestamp	1	8
ts_{exp}	expiration time of the allocation	1	4
HA	hop authenticators	λ	3

Transmission-phase bandwidth efficiency. GLWP is intended to complement and coexist with best-effort traffic. Isolation between GLWP and best-effort traffic is achieved using queuing disciplines on routers [23]. If the capacity is not fully utilized by GLWP, free capacity can be automatically reallocated to best-effort traffic, such that no bandwidth is wasted.

VII. Implementation and evaluation

To show that GLWP satisfies efficiency requirement E1, we implemented the transmission phase procedures for the data plane (§V-B) according to the specification in Algorithm 2.

A. Transmission-phase implementation

Implementation. Our implementations of the GServ of the source AS (transmission phase only) and on-path border routers are based on Intel DPDK [24]. We use a secure MAC construction for hop keys and hop authenticators [25], CBC-MAC, with AES-128 as the underlying block cipher. We use the widely-available Intel's AES-NI hardware instructions to speed up AES computations [26].

GLWP packet structure. GLWP transmission packets are encapsulated inside the packets of the hosting network. In our evaluation, GLWP packets contain an Ethernet header, a GLWP header, and some GLWP payload. Apart from the fields described in Algorithm 2, the GLWP header also contains the number of on-path ASes (λ) and a hop pointer (HP), that allows routers to locate the hop authenticator in the header at forwarding time. The lengths of the path and hop authenticator fields are dynamic and grow with the number of ASes on the path. The GLWP header fields are described in Table I.

Measurement setup. Our evaluation testbed consists of a Spirent SPT-N4U, which serves as a packet generator and bandwidth monitor, and a commodity server with an 18-core Intel Xeon 2.1 GHz processor. The latter runs GLWP as either the source router or an on-path router. The two machines are connected with a 40 Gbps Ethernet link. We measure the throughput (total traffic forwarded) and the goodput (payload-only fraction of traffic) at the bandwidth monitor, and the packet-processing time at the server.

Evaluation metrics. We evaluate the influence on performance of (i) GLWP payload size (p), (ii) number of CPU cores dedicated to packet processing, and (iii) number of on-path

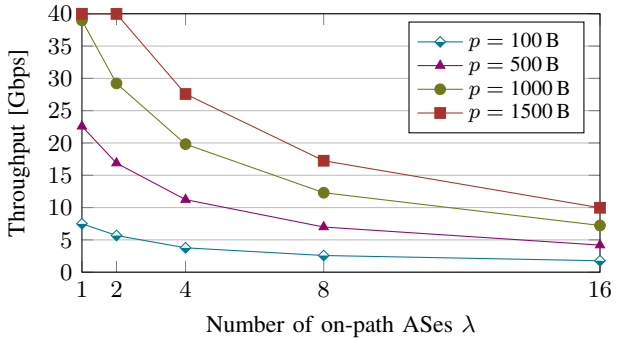


Figure 2: *GLWP packet throughput generated on a single core of the source AS's GServ, for different GLWP payload sizes.*

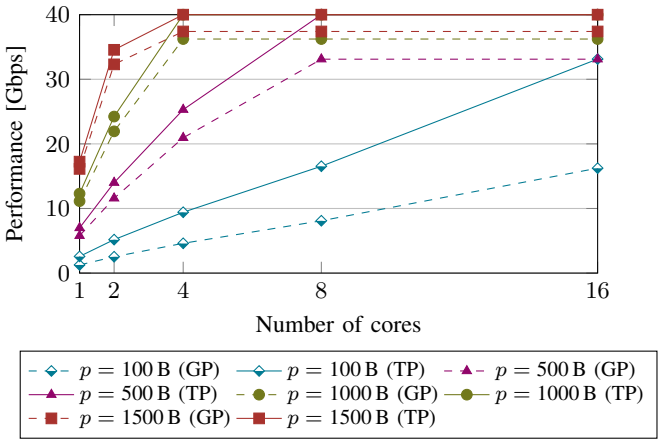


Figure 3: *Throughput (TP) and goodput (GP) of the GServ of the source AS as a function of the number of cores, for different GLWP payload sizes, and for 8 on-path ASes.*

ASes (λ). Because the length of the GLWP header depends on λ, the maximal GLWP payload that fits into a standard Ethernet packet also depends on λ. To allow for measurements with a constant but comprehensive payload range, independent of λ, we enable Ethernet jumbo frames (frames with more than 1500 B payload) on both machines.

B. Results

Source AS's GServ. Figure 2 shows the throughput achieved by one CPU core of the source's GServ. Because the overhead caused by the computation of the hop authenticators increases with the number of on-path ASes, the throughput decreases with the path length. Further measurements showed that packet-processing time does not depend on the payload size. Using more cores, the link can be saturated even for packets with minimal payloads (Fig. 3).

On-path border routers. The forwarding performance of a border router of an on-path AS is depicted in Fig. 4. The results show that eight cores are sufficient to saturate the link for all (non-zero) payload sizes. Further evaluation showed that the packet-processing time is around 280 ns, irrespective of the payload size and of the number of on-path ASes. We were able to forward 45 million packets (without payload) per second using only 14 cores. The calculation and expansion of

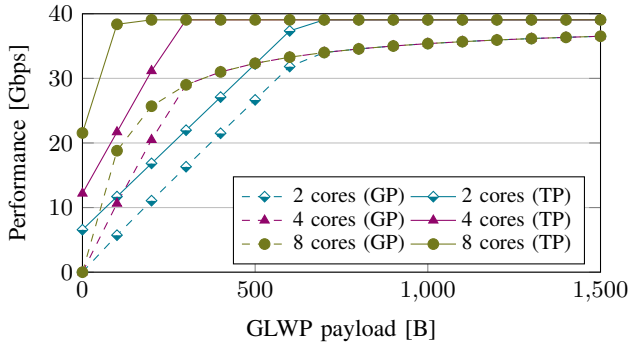


Figure 4: *Throughput (TP) and goodput (GP) of an on-path border router as a function of the GLWP payload for 2, 4, and 8 cores, and for a path consisting of 8 on-path ASes.*

the hop key (Fig. 5) causes the largest overhead. The traffic monitor and the duplicate-suppression system are also causes of overhead for on-path ASes. However, as we argue in §V-D, these additional systems do not pose a threat to scalability.

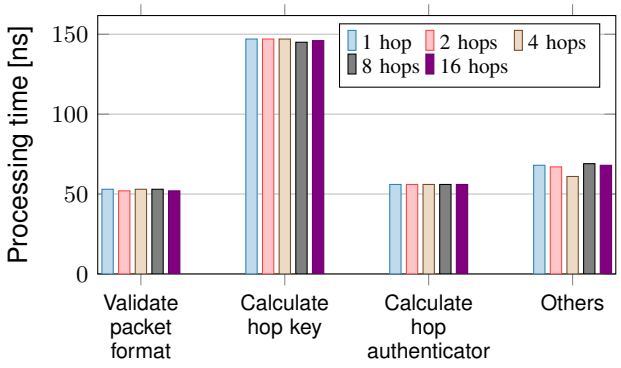


Figure 5: *On-path border router packet-processing times for different sub-tasks. The category 'Others' aggregates all further operations, for example checking that the packet timestamp is current, verifying that the packet was received through the correct interface, or increasing the hop pointer. 'Calculate hop key' also includes the AES key expansion overhead.*

VIII. Related Work

Resource allocation and QoS. Bandwidth guarantees were a central concept of virtual-circuit architectures like ATM [27]. In today's IP-based Internet, the Integrated Services (IntServ) architecture allows to setup bandwidth reservations [28], which can be negotiated through the Resource Reservation Protocol (RSVP) [29]. Due to its high reliance on in-network state, IntServ has never seen widespread adoption.

A wide range of traffic-engineering systems can provide QoS guarantees in intra-domain contexts, such as MPLS [12] with OSPF-TE [30] or SDN-based solutions [31]. However, in contrast to GLWP, which supports autonomous nodes, all these systems require a central controller. Differentiated Services (DiffServ) [32] uses the prioritization of certain types of traffic to achieve QoS objectives. However, DiffServ does not provide any bandwidth guarantees or authentication and is thus also limited to intra-domain contexts.

Capability-based protocols. Following Anderson et al. [33], several mechanisms use cryptographic tokens to express capabilities and counter (D)DoS attacks, including SIFF [34], NetFence [35], and CoDef [36]. However, these systems only convey information about which flows are desired by the destination, and do not perform network resource allocation. In contrast, GLWP provides a ready-to use allocation algorithm.

Based on future Internet architectures, the bandwidth-reservation architectures STRIDE [37] and SIBRA [38] have been proposed recently. While increasing scalability compared to previous systems like IntServ, they fundamentally rely on concepts of future architectures and require a substantially more complicated reservation setup compared to GLWP.

IX. Discussion and conclusion

With the emergence of safety-critical applications, Internet communication has become systemically important to deliver a high level of availability. We leverage the GMA algorithm [2] to propose the GMA-based light-weight communication protocol (GLWP), a concrete approach for achieving high-availability low-rate communication in the Internet. GMA allows each node to define its own neighbor-based policies, which give rise to implicit and sustainable *global* allocations.

Inspired by previous capability-based protocols, GLWP enables scalable bandwidth guarantees even in an adversarial environment *without requiring per-path or per-connection state at the ASes*. It can thus scale to Internet-size networks, while its efficient forwarding process makes it suitable for multi-Gbps network links. In addition to achieving high performance, GLWP can also withstand malicious actions against both the reservation setup and the forwarding process. As GLWP is a distributed protocol, its bandwidth allocations are particularly relevant for inter-domain routing, where central control is inherently impossible and dedicated network infrastructure is prohibitively expensive. Thanks to the unique blend of these three characteristics—scalability, security, and decentralization—GLWP is perfectly positioned to provide QoS protection to critical-yet-frugal traffic.

References

[1] M. Arnold, "Ripple and Swift slug it out over cross-border payments," https://www.ft.com/content/631af8cc-47cc-11e8-8c77-ff51caedcde6, 2018.

[2] G. Giuliari, M. Wyss, M. Legner, and A. Perrig, "GMA: A pareto optimal distributed resource-allocation algorithm," in *SIROCCO*, 2021, https://arxiv.org/abs/2102.10314.

[3] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains," in *Financial Cryptography and Data Security*. Springer, 2016.

[4] Tradeblock, "Analysis of bitcoin transaction size trends," https://tradeblock.com/bitcoin/historical/1w-f-tsize_per_avg-01101, 2015.

[5] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *IEEE Symposium on Security and Privacy*, 2017.

[6] The Diem Association, "Diem white paper v2.0," https://www.diem.com/en-us/white-paper/, 2020.

[7] S. Dawkins, "Path Aware Networking: Obstacles to Deployment (A Bestiary of Roads Not Taken)," https://datatracker.ietf.org/doc/html/draft-irtf-panrg-what-not-to-do-17, Internet-Draft, 2021.

[8] K. Kilkki and B. Finley, "In search of lost QoS," https://arxiv.org/abs/1901.06867, 2019.

[9] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, 1983.

[10] P. Tune and M. Roughan, *Recent Advances in Networking*. SIGCOMM eBook, Aug. 2013, vol. 1, ch. Internet Traffic Matrices: A Primer.

[11] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, 2014.

[12] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031, 2001.

[13] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, 2018.

[14] A. Perrig, P. Szalachowski, R. M. Reischuk, and L. Chuat, *SCION: A Secure Internet Architecture*. Springer, 2017.

[15] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: secure and adoptable source authentication," in *USENIX NSDI*, 2008.

[16] B. Rothenberger, D. Roos, M. Legner, and A. Perrig, "PISKES: Pragmatic Internet-scale key-establishment system," in *ASIACCS*, 2020.

[17] J. Kohl and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, 1993.

[18] T. Lee, C. Pappas, A. Perrig, V. Gligor, and Y.-C. Hu, "The case for in-network replay suppression," in *ASIACCS*, 2017.

[19] H. Wu, H.-C. Hsiao, and Y.-C. Hu, "Efficient large flow detection over arbitrary windows: An algorithm exact outside an ambiguity region," in *IMC*, 2014.

[20] J. Katz and A. Y. Lindell, "Aggregate message authentication codes," in *Topics in Cryptology – CT-RSA*. Springer, 2008.

[21] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra, "Verifying and enforcing network paths with ICING," in *ACM CoNEXT*, 2011.

[22] T. Böttger, G. Antichi, E. L. Fernandes, R. di Lallo, M. Bruyere, S. Uhlig, G. Tyson, and I. Castro, "Shaping the Internet: 10 years of IXP growth," https://arxiv.org/abs/1810.10963v3, 2019.

[23] T. Velmurugan, H. Chandra, and S. Balaji, "Comparison of queuing disciplines for differentiated services using OPNET," in *International Conference on Advances in Recent Technologies in Communication and Computing*, 2009.

[24] DPDK Project, "Data Plane Development Kit," https://dpdk.org, 2020.

[25] M. Bellare, J. Kilian, and P. Rogaway, "The security of the cipher block chaining message authentication code," *Journal of Computer and System Sciences*, vol. 61, no. 3, 2000.

[26] J. Rott, "Intel advanced encryption standard instructions (AES-NI)," *Technical Report, Intel*, 2010.

[27] H. Saitō, *Teletraffic Technologies in ATM Networks*. Artech House, 1994.

[28] R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC 1633, 1994.

[29] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," RFC 2205, 1997.

[30] D. Katz, K. Kompella, and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2," RFC 3630, 2003.

[31] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho, and C. Yang, "Traffic engineering in software-defined networking: Measurement and management," *IEEE Access*, vol. 4, 2016.

[32] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, 1998.

[33] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet denial-of-service with capabilities," *ACM CCR*, vol. 34, no. 1, 2004.

[34] A. Yaar, A. Perrig, and D. Song, "SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks," in *IEEE Symposium on Security and Privacy*, 2004.

[35] X. Liu, X. Yang, and Y. Xia, "NetFence: preventing internet denial of service from inside out," *ACM CCR*, vol. 40, no. 4, 2010.

[36] S. B. Lee, M. S. Kang, and V. D. Gligor, "CoDef: Collaborative defense against large-scale link-flooding attacks," in *ACM CoNEXT*, 2013.

[37] H.-C. Hsiao, T. H.-J. Kim, S. B. Lee, X. Zhang, S. Yoo, V. Gligor, and A. Perrig, "STRIDE: Sanctuary trail – refuge from internet DDoS entrapment," in *ASIACCS*, 2013.

[38] C. Basescu, R. M. Reischuk, P. Szalachowski, A. Perrig, Y. Zhang, H.-C. Hsiao, A. Kubota, and J. Urakawa, "SIBRA: Scalable Internet bandwidth reservation architecture," in *NDSS*, 2016.