Service-based Forwarding via Programmable Dataplanes

René Glebke*, Dirk Trossen†, Ike Kunze*, David Lou†, Jan Rüth*, Mirko Stoffers*, Klaus Wehrle*
*Communication and Distributed Systems, RWTH Aachen University, Aachen, Germany
{glebke, kunze, rueth, stoffers, wehrle}@comsys.rwth-aachen.de
†Huawei Technologies Germany, Düsseldorf, Germany
{dirk.trossen, zhe.lou}@huawei.com

***Abstract*—Access to networks for purposes of executing remote services has become a dominant form of communication, while virtualization has enabled the flexible and fast deployment of those services across distributed locations. This seems at odds with the design that drives the network layer of the Internet. Our paper presents an approach to flexibly deploy a network layer solution for optimizing service access within a single domain, while retaining full connectivity to Internet-based services. We discuss design considerations and the resulting design. We analyze expected gains from our solution.**

***Index Terms*—programmable forwarding, semantic addressing, load balancing**

I. Introduction

Data being sent to remote servers for processing is a common pattern that can be found in many communication domains, such as in the public Internet but also in industrial networking. Furthermore, it is estimated that approximately 72% of all data center traffic stays within the local domain, i.e., terminates in the center where it originated [1].

In addition, compute resources can be provisioned quickly through virtualization technologies. This leads to an increasing replication of service functionality in what is called *service instances* in the following, enabling the optimization of service experience and quality, with an impact on a number of domains such as industrial networking [2] or immersive gaming [3].

From those observations, we can define the goal for a communication infrastructure as one focusing on the WHAT of the communication, while regarding the place of execution, the WHERE, as ephemeral. This goal is seemingly at odds with existing Internet technologies, which route information based on locator identifiers, while using mapping services, such as the Domain Name System (DNS), to determine the location.

Efforts such as information-centric networking (ICN) [4], [5] have similarly identified the purpose of communication as key to developing a network layer, but ambitions of globally replacing IP routing have stood in the way of widespread deployment so far. In our approach, we rely on two key aspects for realizing a service-oriented communication infrastructure that would avoid a similar fate. The first is the concept of a *limited domain* [6], which delimits the development and deployment of technologies to the scope of network and endpoint behavior that is specific to that domain. This means that we see our solution limited to a single domain, while enabling full access to non-domain local services via the existing Internet. We also see our solution deployed alongside (and utilizing, where needed) the locator-based addressing (and routing) of existing Internet solutions, addressing the challenge of *backward compatibility* but also *interconnection* to existing services. Deployment examples include industrial networks with services executed on "workers" to aid equipment control.

The second aspect is the emergence of *programmable forwarding planes*, i.e., the capability to change forwarding behavior within the limited domain through programmability that may exist in a limited number or all of the nodes. Technologies such as P4 [7], [8] enable flexible but still high performing execution of those programmable forwarding instructions, thereby addressing the *performance* challenge that often plagues initial deployments that aim to drive adoption. Such programmability also opens up the opportunity to tailor the network's behaviors specific to (sets of) services. Through this, our solution addresses the challenge of *fast deployment* to avoid the fate of ICN and similar propositions, while relying on orchestration and endpoint programmability to realize necessary changes to endpoint and network behavior.

We see three opportunities for our solution. First, we expect an impact on *network latency* since communication can directly take place between the relevant endpoints, removing the need for any reflection points, such as brokers and alike. Second, we expect the *service latency* to improve since the decision which instance of a service should serve a request can be realized at the network level, not requiring, e.g., application-level load balancers such as those employed in content delivery networks (CDNs). Last, we also expect to improve *deployability* through combining the programmability of forwarding planes with the general trend of service orchestration and application-centric development in end systems.

We organize the remainder of this paper as follows:

- We discuss design considerations, taken into account in our work, in Section II.
- We then shape those considerations into a system design in Section III for realization in a limited domain.
- We analyze the impact that our solution may have on the aforementioned key aspects, in Section IV.
- We conclude our paper, including a discussion on possible extensions of this work, in Section V.

II. Design Considerations

Our main design goals are the *routing on service names* and the *steering of traffic* aided by in-network programmability, while limiting the deployment to *limited domains* that strongly exhibit the expected service-oriented nature of communication. We elaborate on the main considerations to achieve these goals through our design in the following.

A. Addressing

We observed already in the introduction that the service-centricity of the endpoint behavior is key, which is directly reflected in the addressing utilized. Here, the identity of the service matters foremost in finding suitable service instances out of the pool of (many) possible ones. For this, we utilize the concept of a *serviceID* that represents the overall service that is implemented by those possible service instances.

A number of choices exist for the naming of services. A first option is to use structured binary names, such as those used in ICN [9]. Another option is that of a *P:L* name, where *P* is rooted in the security credentials of the principal owner of the services and the label *L* identifies the service itself [5]. Both approaches, however, were designed with global deployment in mind, resulting in possibly long names, although [9] does allow for app-specific short names. Given the limited domain nature of our envisioned deployment, we advocate instead to use simple binary identifiers of fixed size, assuming the existence of a domain-local service for mapping app-level names onto those identifies. Using such shorter, fixed size identifiers also allows for a practical integration of the identifier into existing packet structures.

Apart from the service identifier, the location of the service instance does still matter, as we discuss in more detail in Section II-C. For that reason, our design also utilizes IP addresses (and therefore IP routing and forwarding solutions).

B. Service Scheduling

When selecting the most 'suitable' service instance from the pool of deployed ones, several criteria may define what is 'suitable'. Traditional constrained routing solutions, such as EIGRP [10], utilize network-centric metrics, foremost delay and bandwidth, to steer traffic to a specific destination. We instead foresee *service-specific metrics*, such as the load of an instance, certain compute capabilities (that may differentiate the service implementation in terms of execution speed or storage capability), or end-to-end delay (which goes beyond network delay to include, e.g., the processing for a response). Hence, our design needs to support such service-specific metrics and the necessary selection criteria to choose based on such metrics. We here expect the biggest impact of using programmable dataplanes through efficiently implementing those metric-specific forwarding decisions, while we still see those service-specific decisions as being aligned with end-to-end arguments, where the programmable forwarding behavior is seen as an *"...incomplete version of the function [standing at the endpoints] provided by the communication system [that] may be useful as a performance enhancement."* [11].

C. Instance Affinity

While the identification of the most suitable service instance is driven by the serviceID and the metric to choose one of the available service instances, our design must also accommodate the situation in which traffic must continue to be steered to an initially selected service instance. Reasons for this situation are, e.g., the creation of application-specific state at the service instance; we call this property *instance affinity*.

For this, we utilize existing locator-based addressing to support instance affinity by differentiating the initial *service request*, forwarded based on the serviceID/metric selection, from an *instance request* to a specific instance. Here, the response from the service instance chosen in the service request provides the necessary locator information to realize the continued communication with that specific service instance.

It is important to understand that only the service instance and client can decide when affinity is no longer required in that the client issues another service request, rather than use the locator of the service instance it maintained affinity to before. This would also support short-lived, even single packet requests by only using service requests.

D. Forwarding Behavior

As discussed before, the forwarding behavior extends existing locator-based forwarding (i.e., longest-prefix match of IP addresses) through a service-centric behavior. Here, the service identifier and metric need to be used to select from a set of available service instances. The information on those service instances is assumed to be provided by a routing protocol to build a suitable *service routing table* alongside the existing IP routing table. In the case of a service request, the set of instances, determined by the serviceID, is then further constrained through the metric-based decision. If instead, an instance request is provided (indicated by the missing serviceID), normal locator-based forwarding applies.

While a design may strive for supporting a rich set of supported metrics, our selection is driven by the ability to realize them in programmable forwarding planes, while allowing for clear impact on service and network latency.

E. Scope of Limited Domain

As discussed before, we see our solution deployed within a limited domain. Coupled with the intention of utilizing programmable forwarding switches, we see the scope of such domains reach from a single switch (e.g., in localized industrial deployments) to larger bridged Ethernet deployments (e.g., in larger factories) to larger routed domains (e.g., campus networks). The extension to the latter requires a routing protocol for distributing service identification and metric information; we discuss this aspect in Section III-B.

A key aspect for the limited domain is its interconnection to other limited domains and the Internet itself[1]. In our current design, we assume client knowledge to differentiate the access

[1]Note that the limited domain itself can span different geographical locations, e.g., different industrial sites, by virtue of tunnels between the separate deployments, still forming a single limited domain.

to services within the limited domain versus those provided in the Internet. An approach for ensuring true interconnection without such client knowledge required is left for future work.

F. Security

Security in service interactions is paramount, both at the level of the service relation as well as content. For the latter, encrypting the payload of a service request is entirely left to the application, similar to the Internet. Technologies such as TLS can be used over a suitable transport protocol like TCP or QUIC. In its current version 1.3 [12], this would facilitate 1-RTT secure transactions. With newer work, such as *Encrypted Client Hello* (ECH)[2], even 0-RTT can be achieved, thereby well supporting single request interactions.

The privacy of the service relation can be achieved by using service-specific serviceIDs (e.g., by providing a mapping service separate from the network provider). If the deployment, however, is entirely vertical, i.e., the service and network provider may be the same entity, privacy of the communication relation may not be an issue, in which case a simple hashing of service information into serviceIDs may suffice.

III. System Design

We now present our design utilizing programmable data-planes (PDPs). Due to the large variety of PDPs, we do not aim for an implementation on a specific PDP but generality. For this, we target the more abstract *RMT* architecture [13], whose match-action pipeline based operation principle is most prominently embodied by the P4 language [7], [8] and implemented by a multitude of PDPs (see e.g., [14] for an overview). Furthermore, we aim for coexistence with legacy systems, in order to enable an incremental deployment.

We first present the major components of our design together with the broad interaction scheme. We then discuss the intended system scope, the addressing mechanism, and the constraint model clients and service instances use to express their desires and abilities. Finally, we show how we realize our forwarding mechanism on PDPs (switches) based on P4/RMT.

A. Architecture

Our system, visualized in Figure 1, comprises three types of entities: workers, clients, and service scheduling units. *Workers* offer a set of services, thereby acting as service instances for those services. A worker can be any type of uniquely identifiable system, such as a hardware server or a virtual machine (already running or instantiated on-demand). Services are accessed by *clients*, whose service requests are dispatched by *service scheduling units* (SSUs) on programmable switches. All entities are interconnected within some limited domain. Services are addressed using a unique *serviceID*; one or more workers may provide realizations of a service exposed by a given serviceID.

Offering services: Workers announce their participation in providing a given service by announcing its associated serviceID, together with constraint information (which we

[2]https://blog.cloudflare.com/encrypted-client-hello/

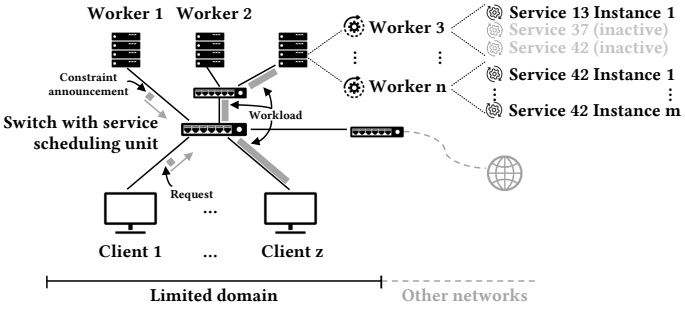

Fig. 1: Main entities and relations in our architecture.

detail in Section III-D), to their local SSU, which in turn disseminates the information to its peer SSUs. For gathering of constraint information, workers employ *service monitors* to ascertain their current constraints for executing instances of the services they offer. Monitors may coordinate with running instances of the service they monitor, as well as with other local monitors, to determine the current constraints. The result of a monitor's work is a *constraint value* (CV) sent to the SSU, which matches the current CVs of all workers against the requirements contained within the requests received by clients. Monitors can run periodically or on an on-demand basis, depending on the nature of their respective service; whenever they detect a change in the CV they deem significant, they inform the SSU accordingly.

Accessing services: When a client wishes to access a service, it sends a *service request* to its local SSU, which in turn schedules the request packet to a chosen worker (described later in Section III-E). The worker then processes the request and, depending on the nature of the service, sends a response to the client. Workers support the execution of *workloads*, representing a stream of packets that a client sends when using a certain service and for which an instance relationship must be maintained for reasons of, e.g., ephemeral state being created at the worker (independent workloads may be dispatched to separate workers). We refer to requests by clients within the same workload following the initial request and a response by the worker as *instance requests*. When responding to the clients, workers provide their IP locators, which the clients use for their subsequent instance requests. The requests from clients also encode means for workers to provide responses that allow clients to distinguish responses to multiple requests issued in parallel (e.g., port numbers in plain TCP/UDP or connection IDs in QUIC headers [15]). Note that while service requests target single services only, workers may likewise request the execution of subsequent services for their workloads, which enables processing chains.

The described interaction between client and workers allows for a 0-RTT communication from the client to the chosen worker. Within a multi-packet workload, however, the client will need to wait for the worker's response first to obtain its IP locator. This, however, is consistent with the typical behavior in connection-based scenarios (e.g., TCP), albeit introducing an initial 1-RTT delay. However, if combined with transport

layer security techniques, such 1-RTT handshakes are part of the security mechanisms anyway. Secure 0-RTT exchanges are possible e.g., via the approaches discussed in Section II-F, but we consider this future work for now.

B. Scope of System

As indicated in Figure 1, we generally allow the entities in our system to be distributed over the limited domain in which our system is deployed . This means that clients and workers may not be directly attached to the switch running the SSU, and also that there may be both multiple paths between clients and workers, as well as multiple SSUs or intermediary switches that do not participate in our system. All service (and instance) requests would then be relayed towards the SSU without the need to change the behavior of intermediate systems, except for an eventual addition of appropriate prefix-based forwarding rules. We assume that the SSU information would be provided to the client and worker through suitable discovery protocols, such as DHCP.

This assumption leads us to an initial centralized scope of the limited domain system, with a single SSU dispatching incoming service requests. However, relying on a single SSU will lead to non-optimal paths since placing such SSU optimally in regards to all entities would require networks of small size. Hence, we foresee that multiple SSUs exist, possibly at every ingress point to the network.

This in turn requires that serviceIDs and constraint information, announced by workers to a specific SSU, are distributed to all other SSUs in the system. Such distribution of *routing information* could be realized using existing protocols and strategies such as OSPF [16], [17]. Depending on the network size, the nature of the constraints (e.g., those requiring frequent updates), as well as the number of services offered by workers, a flooding-based routing approach will not scale well. Fortunately, announcing constraints (and their operations) provides the opportunity for aggregation and suppression of announcements. For instance, considering a *min* operation on the announced CVs for some service, the announcement of *CV=5* after another worker already announced *CV=4* to the same SSU does not change the minimum of announced CVs and needs hence not be forwarded to other SSUs. The development of a routing protocol that would utilize such capability, however, is beyond the scope of this paper.

C. Addressing

We utilize simple opaque binary identifiers of fixed size (unsigned integers) for our serviceIDs to limit the possible overhead in service requests and ease deployment of our solution. As those serviceIDs are opaque, client and service developers need to negotiate which serviceIDs to employ so that the service can be reached, while avoiding ID collisions with other services. Thus, either some entity within the limited domain needs to specify the serviceIDs to be used by clients and servers, or existing discovery/mapping mechanisms such as DHCP/BOOTP [18] or DNS can be employed, in turn relying on some assignment or mapping

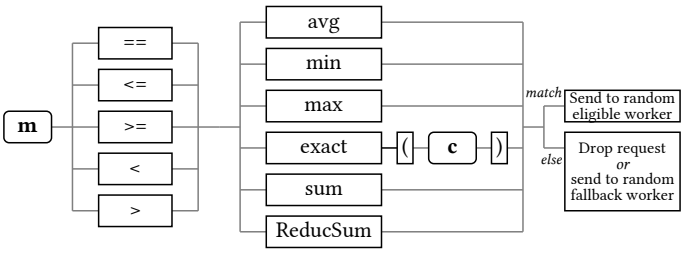

Fig. 2: Constraint model for selecting service instances. Client demands (m, left) are matched against worker supply (right).

services to provide the right serviceID. The nature of our system does not prescribe lookups of serviceIDs to be repeated as long as the requested service does not change by some administrative or programmatic necessity, and hence these negotiation mechanisms constitute a negligible overhead.

This allows us to integrate requests from clients into the majority of existing system architectures. When employed within an IP-based realm, serviceIDs can e.g., be assigned per virtual subnets (as in CIDR [19]), with the SSU acting as the gateway into these subnets. The client then utilizes the IP address of the worker to which the service request was initially sent for issuing subsequent instance requests.

D. Constraint Model

Workers and clients have service-specific *constraints* they impose on instances and workloads, e.g., the time they require for executing the requested functionality. We model this as matching the *demand* of clients (included in their request packets) against the *supply* from workers (announced as CVs) at the SSU. To simplify the scheduling process, we assume that the constraints on the client side do not change while a workload is being processed, i.e., clients determine their constraints at the moment they send their service request to the SSU, while for any subsequent packet of the same workload, the same worker is being used (i.e., instance affinity is maintained). Clients can declare new demands by issuing a new service request. We also leave handling changes in worker constraints during an ongoing workload to the involved applications since they are best positioned to do so.

Figure 2 shows the supported constraint-based operations. We allow both equality and relative comparisons of a client's demand *m* against either *exact* advertised CVs *c*, as well as against some aggregated values over all advertised CVs for a respective serviceID, specifically *avg*, *min*, *max*, the *sum* or the *reducing sum*. For the latter, the sum of the advertised constraints is reduced by the value *m* before forwarding, while new arriving CV values increase sum, which in combination allows for consumption-based scenarios. Our model also supports expressions like *choose the min/max over all advertised CVs* when a client sends the wildcard value $\star$ in its service request. For each request, any worker with a CV satisfying the constraints of the client is a potential recipient.

As examples, the request "$m \leq 5$" matches all workers with a CV ≤ 5, "$\star == min$" matches all workers that have

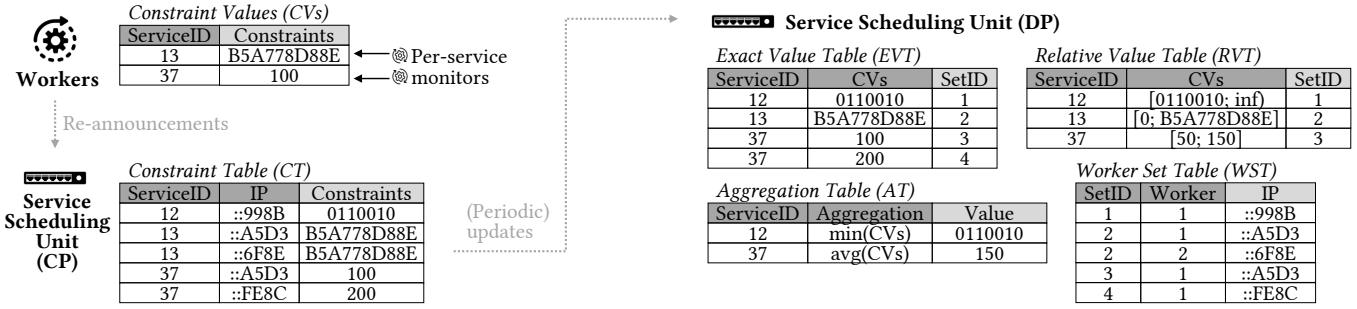

Fig. 3: Based on announcements from service monitors entities on the workers (left), the control plane of the service scheduling unit generates views of the current constraints (right) which allow the dataplane to schedule requests by clients to appropriate workers. (The figure shows the full information contained within EVT for this example, and excerpts for the other tables.)

advertised the current minimum CV, and "$m \geq min$" matches any worker with the minimum CV as before, but only if that minimum is less or equal to m. Clients can instruct the SSU to either drop a request that cannot be satisfied (the SSU then informs the client using e.g., an ICMP message) or to choose a random worker as a fallback.

Note that metrics that contribute to constraints of a worker can differ largely between services and their exact meaning is determined only by the associated service. It is not necessary for other services or the SSU to understand the semantics that define the CV nor its operations. Our only requirement for clients' m values and worker's CVs is for them being fixed size unsigned integers. This eases implementation of the matching logic on the SSU, described in the next subsection.

We position our approach here as a baseline, supporting a single constraint value per service. However, our approach can be extended to provide a choice between more than one constraint, so that constraints which are hard to combine into one (e.g., worker latency vs. worker throughput) can be separately used. In this case, workers/clients would need to specify which *constraintID* they are targeting in their announcements/requests, and the tables in the SSU need to be amended by a column for the constraintID. However, *multi-dimensional* constraints are not supported since, as we will see in the following, restrictions concerning lookups and the handling of sets on the dataplane render multi-dimensional lookups on current PDPs very delicate without heavily involving the control plane in each decision. We therefore defer the support for such complex constraints to future work.

E. Scheduling Service Requests

To implement our scheduling operations directly within the dataplane (DP), we utilize a combination of comparisons and table lookups. However, the capabilities of P4/RMT DPs with regard to table lookups are limited. Most notably, the structure of lookup keys (i.e., which columns to search) is predetermined at compile time, and tables may be accessed at most once for a packet passing through the match-action pipeline, making programmatic iterations through tables to find the best match impossible. Furthermore, lookups expressing arithmetic relations (e.g., "$key \geq x$") are not possible. Range lookups ("$key \in [a, b]$") are possible, but only the CP can define the bounds while the DP provides the key. In the following, we detail how we still achieve the desired variety of matchings between clients' requests and workers' offers.

Data structures: An overview of the major data structures involved in our approach is presented in Figure 3. First, all CVs received from workers are handed to the CP (left part of the figure) to save the contents in a *constraints table* (CT). To enable lookups on the DP, the CP then fills four tables accessible by the DP: *(1)* the *exact value table* (EVT), which groups workers with the same CVs together; *(2)* the *relative value table* (RVT), which groups workers of similar CVs together into buckets and employs P4/RMT *range* lookups[3]; *(3)* the *aggregation table* (AT), which provides meta-information on the received CVs (such as minimum, maximum, and average)[4], and at last *(4)* the *worker set table* (WST), which stores the groups to which workers have been assigned in the EVT and RVT. The keys to ECT, RVT and AT are composed of the respective serviceID as well as the CV and/or an ID of the requested aggregated information (we address the WST later). This allows us to store all necessary values for all services within the same four structures.

Handling constraint updates from workers: The tables accessible to the DP are updated by the CP when a new CV is received from a worker. The required operations for updating known workers are straightforward: A new CV simply means replacing a single old entry in the EVT, touching the previous and new buckets/groups in the RVT/WST, and performing some minor calculations with the help of the CT to update the meta-values in the AT. Most PDPs support changes to a limited number of entries without interruption of the DP, and we expect modern CP implementations to be capable of handling hundreds of updates per second at least. In case the update rate of workers is high, the CP can also first use its CT to collect new announcements for a certain period, and

[3]The EVT can also be represented within the RVT when ranges always comprise exactly one value, but often, PDPs put limits on the number of ranges in tables, so that we chose two distinct tables for increased scalability.

[4]The reducing sum additionally requires P4/RMT *registers*, which can be read and written from both CP and DP. We leave out the details and treat it like the other aggregation types in the following for simplicity.

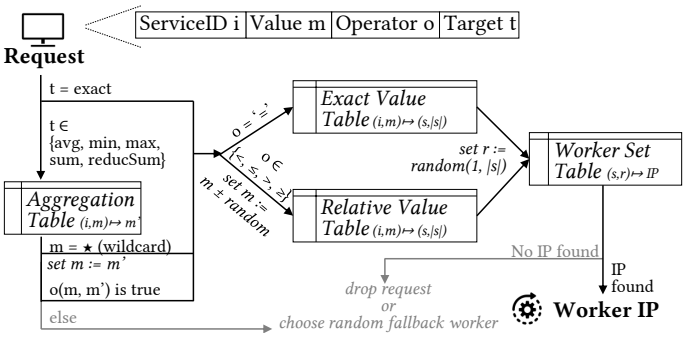

Fig. 4: Depending on the value, operator and comparison target in a request, the service scheduling unit uses different sequences of table lookups, comparisons and lookup value rewrites. A final semi-random lookup in the worker set table yields the IP of a worker able to satisfy the request.

then perform bulk updates of the tables in the DP. Our design in principle also allows that services and the SSU negotiate individual update rates, which would require coordinating with the authority managing the SSU; we defer this to future work.

Matching requests and offers: Finding a matching worker for a client's request employs a sequence of lookups in the AT, EVT/RVT and WST against the constraint value sent by the client. As visible in Figure 4, the sequence differs depending on the type of comparison the client wishes the SSU to fulfill. While equality comparisons using the EVT are straightforward (the value contained within a client's request is directly used as part of the key), relative comparisons via the RVT require a different approach. As P4/RMT only allows matching against a single (exact) value in a table, we need to transform the request of the client into a format that allows us to emulate a relative lookup. We accomplish this by deriving a new lookup value based on the constraint value received from the client and the chosen relation. Our approach is based on the observation that when a client requests some worker satisfying a relation, then *any* worker with an announced CV to the left ($\leq, <$) or the right ($\geq, >$) of the value sent by the client will do the job. We consequently shift the received constraint value into the required direction, by subtracting or adding a random value[5] and capping the results at 0 and the maximum value for CVs, respectively (to stay within the bounds of the requested relation). The resulting value can then be used as the lookup value representing the requested relation in the RVT, and the random offset ensures a distribution of requests among the workers that fall into the value range of the relation. Note that the bounds of the ranges in the RVT need to cover the entire domain of possible CVs to guarantee that the lookups succeed; the maintenance overhead for the CP is proportional to the number of different CVs advertised by a service.

As indicated in the lower part of Figure 4, requests involving the meta-information contained within the AT require a lookup that precedes those in the EVT or RVT. By first performing a lookup on the respective meta-information in the AT, we can use the resulting value in lieu of a concrete value received from the client in the subsequent lookups in the other two tables. This on the one hand enables queries such as "$any\ worker \geq avg$" (when following the first branch below the AT in the figure). On the other hand, it also allows us to compare the client's value against the aggregated value and only continue the scheduling process in case the comparison operation evaluates to *true* (second branch). This comparison can be expressed using unsigned integer operations, which are available on all but the most restrictive PDPs.

Choosing an eligible worker: Lookups to the EVT or RVT yield a set of workers all capable of satisfying a client's request, as well as the size of that set. P4/RMT requires sets to be represented via tables (sets are structures of varying lengths, which are not supported by the majority of current PDPs), and our lookup for a client's request is thus concluded by choosing an appropriate worker by performing a lookup in the WST. For this, we take the resulting worker set ID returned from the EVT/RVT lookup, and complete the lookup key by using a random number as the second part, capping the random numbers so that they do not surpass the size of the worker set. The result of this lookup (shown in the lower right of Figure 4) is the address of a unique worker capable of servicing the client's request, and we conclude our scheduling by forwarding the client's request accordingly. In case no suitable worker set is found and the client's request thus cannot be satisfied, the SSU either drops the request (informing the client accordingly) or chooses a random fallback worker among those generally available for the requested serviceID. Both actions require some additional operations (i.e., sending a failure message to the client and looking up the fallback worker, respectively), which are not detailed here for simplicity, but which can also be entirely executed on the DP.

IV. Analysis

We now analyze different aspects of our design, starting with security, then the impact on network latency compared to application-level solutions as well as the impact on service completion time compared to traditional approaches. We conclude with outlining the opportunities brought about by the realization of our mechanisms on programmable dataplanes.

A. Security

Authentication: Clients and workers can continue to apply established authentication and validation mechanisms (e.g., PKI/X.509), as long as a notion of the serviceID used by the client to initially access the service is included within that process. Furthermore, constraint values sent from service monitors towards the SSU can be encrypted and authenticated as long as the control plane of the SSU has cryptographic capabilities, so that neither a spoofing of service states nor a participation as an unsolicited worker is possible in our system.

Security of discovery: As we utilize mechanisms, such as the DNS or other mapping services, for the management and

[5]Random number generators are not defined by P4/RMT *per se*, but most PDPs include one for use on the DP.

discovery of available services and serviceIDs, the security of this part of our design is equal to that of these mechanisms.

Endpoint security: As we expect the workers and installed service applications to be benign (since the limited domain controls their setup), there are also no adverse effects conceivable from running multiple services on the same machine.

Dataplane security: As most PDPs currently do not have cryptographic capabilities on the dataplane, requests from clients for the mean time cannot be encrypted or authenticated, so that fingerprinting and eavesdropping attacks on serviceIDs and the requested capabilities of workers by clients are possible. However, this restriction is not imposed by our design; once cryptography becomes possible on the dataplane, a suitable decryption and authenticity validation (or a check for access rights of the client to the requested service) can preclude the other scheduling steps. Until such functionality is largely available, the opaqueness of serviceIDs and CVs at least hampers eavesdroppers from gaining instant knowledge.

B. Network latency

To capture the impact of network latency on our design versus traditional approaches, we investigate the differences in Figure 5. Here, we model a simple network in SimPy[6] using different latency distributions and show the combined latency of 10 000 random samples. The network consists of a low latency wireless link (e.g., 5G) followed by a central router at which a directory service, a load-balancer, and all actual services are connected. We model the wireless link with a constant delay of 5ms plus an exponentially distributed delay with a mean of 2.5ms, the rest are modeled by normal distributions. Our model assumes an average 1ms of delay to pass through the router to any connected entity and 500µs to perform any action (e.g., lookup) both with a very small variance. We compare the time it takes a client to receive a first meaningful byte from a service with our service-based routing employed at the central router to a traditional model that first requests a mapping to a service from a central directory (e.g., DNS), waits for the result and subsequently requests the service. Additionally, we show the impact of having to go through another indirection step after the directory discovery when an application-load balancer is involved; both cases can be handled by service-based routing.

What the figure shows and what is expected is that when employing service-based routing at the central router, we save additional roundtrips. Here, latency gains for the service-based routing primarily stem from not having to traverse the wireless link as often as the traditional approaches. While these gains of course diminish over the course of the duration of a workload, they are especially important for short-lived encounters.

C. Service latency

For the impact on service latency when utilizing a distributed set of workers, we consider the following scenario. A number n of clients send service requests to a number k of workers. We assume for simplicity that the clients are all connected to the same SSU.

[6]https://simpy.readthedocs.io

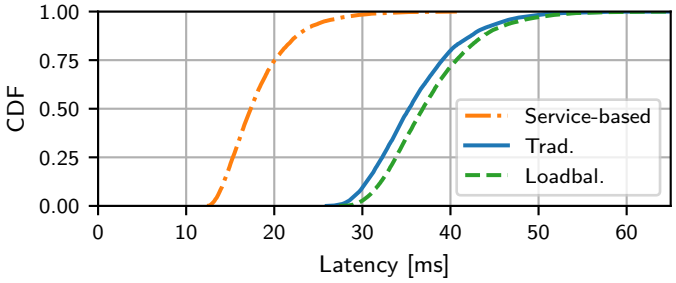

Fig. 5: Expected gains for network latency. Simulation results show that service-based forwarding can significantly cut time to first byte retrieval latencies.

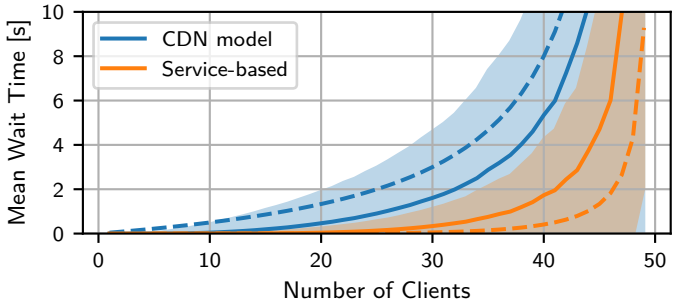

Fig. 6: Mean wait times for CDN and service-based access model with increasing number of clients. Solid lines are simulated results, with shaded areas as standard deviation. Dashed lines are analytical results.

We distinguish two affinity models. In the first, a random server is selected and the client maintains affinity throughout the scenario. This is reminiscent of situations where clients are assigned CDN servers, e.g., based on an initial load balancing decision, and remain connected to the respective servers for the entire workload. We hence call this the *CDN model*. In our second model, we fully utilize the routing level decision to send a service request to any worker, i.e., the affinity here is that of a single request; we term this model *service-based*. A typical use case that would exhibit such behavior is the retrieval of chunk-based data from workers, such as graphic or video assets in gaming or over-the-top video scenarios.

To ground our simulation results, we abstract the first model through an *M/M/1* queueing model with *n/k* clients accessing the single server, i.e., clients will be distributed equally across all available servers (but continue to be served from the assigned one). Our service-based model can be abstracted through an *M/M/k* queueing model where all n clients will access the k servers of the system.

Figure 6 shows the results for our simulation with the solid lines showing the mean waiting time for a total of 1 million requests coming from an increasing number of clients, together with its standard deviation (shaded area). We utilize an inter-arrival time for the packet requests of *10s* by each client and

a service time of *2s* at each server in the system, while setting the number of servers to *k=10*. Additionally, the dashed lines show the analytical results for the aforementioned queueing model abstractions. As confirmed by our analytical model, the service-based access completes service requests much faster than the CDN model, while also providing smaller variances; both are important factors in the overall user experience for the underlying service.

D. Programmability

The work in [3] outlines the combination of mobile application installation with the deployment of micro-services, which are part of the application, through existing service orchestration frameworks like ETSI NFV[7]. We see our solution here complementing this view of future app-centric deployments since the programmability at the level of the forwarding plane enables the insertion of forwarding rules "on-the-fly", all encoded in suitable orchestration templates.

Similar to [3], we envision suitable interfaces to be offered by orchestration platforms, which could in turn deploy the suitable forwarding rules, too, therefore establishing the table structures outlined in Section III-E in intermediary switches. The key point to ensure that rules do not collide across different applications and services is the *serviceID*, i.e., how to generate or obtain such serviceIDs is crucial to ensure the on-demand deployment of any service solution.

V. Conclusions & Future Work

This paper presents a design for a communication solution that improves on service access in a limited domain, utilizing the programmability of its underlying forwarding plane for key capabilities of our design. It outlines key considerations for our design as well as the feasibility of this design within existing frameworks such as P4. Our analysis shows initial results for the expected gains of our solution in terms of network and service latency. Apart from a more thorough platform-based analysis of key performance aspects, there are a number of design aspects that we see need further study.

Routing: To scale in network size, number of offered services as well as frequency of constraint announcements, suitable routing protocols will be required. While link state approaches could be used, as discussed in Section III-B, those do not provide the scale needed due to the excessive flooding. Two key aspects seem promising to reduce the message overhead, namely (i) suppression of announcements along paths where previously announced metrics are already met and (ii) avoidance of loops through suitable construction of virtual loop-free topologies. Both aspects are currently flowing into the development of such routing protocols.

Mobility: A moving client or service endpoint should be supported by our system. Given that endpoints may issue new service requests at any time (even after a mobility event), the only issue may arise when moving during an ongoing affinity relation with a specific service instance. Since the endpoint communicates with the service instance via its IP address, Mobile IP or QUIC sessions could be applied to provide session continuity. For the case of moving service endpoints, the key issue will be to deal with any application state that needs transferring, which is outside the scope of our system.

More advanced service scheduling: Our supported metric operations are relatively simplistic in order to be directly realized in a programmable forwarding fabric. Likely future advances in that programmability will lead to increasingly more complex metric operations being possible to be performed, which in turn open up the opportunity for more advanced service scheduling solutions to be realized. Conversely, the reliance of those more complex service scheduling solutions and their potential benefits for overall system performance may serve as a driver for improving the capabilities of the programmability capabilities we utilize in our design.

[7]https://www.etsi.org/technologies/nfv

References

[1] Cisco, "Cisco Global Cloud Index: Forecast and Methodology, 2016 - 2021," Tech. Rep., 2018.

[2] C. Pallasch *et al.*, "Edge Powered Industrial Control: Concept for Combining Cloud and Automation Technologies," in *IEEE EDGE*, 2018.

[3] I. Kunze *et al.*, "Use Cases for In-Network Computing," IETF, Internet-Draft, Feb. 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-coinrg-use-cases/

[4] V. Jacobson *et al.*, "Networking named content," in *ACM Conext*, 2009.

[5] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM*, 2007.

[6] B. Carpenter and B. Liu, "Limited Domains and Internet Protocols," IETF, RFC 8799, Jul. 2020. [Online]. Available: http://tools.ietf.org/rfc/rfc8799.txt

[7] P. Bosshart *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM CCR*, vol. 44, no. 3, pp. 87–95, 2014.

[8] The P4 Language Consortium, "$P4_{16}$ Language Specification version 1.2.1," 06 2020. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.1.html

[9] M. Mosko *et al.*, "Content-Centric Networking (CCNx) Messages in TLV Format," IETF, RFC 8609, Jul. 2019. [Online]. Available: http://tools.ietf.org/rfc/rfc8609.txt

[10] D. Savage *et al.*, "Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)," IETF, RFC 7868, May 2016. [Online]. Available: http://tools.ietf.org/rfc/rfc7868.txt

[11] J. H. Saltzer *et al.*, "End-to-End Arguments in System Design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, p. 277–288, Nov. 1984.

[12] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," IETF, RFC 8446, Aug. 2018. [Online]. Available: http://tools.ietf.org/rfc/rfc8446.txt

[13] P. Bosshart *et al.*, "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN," in *ACM SIGCOMM*, 2013.

[14] F. Hauser *et al.*, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," 2021, arXiv preprint. [Online]. Available: https://arxiv.org/abs/2101.10632

[15] E. J. Iyengar and E. M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," IETF, Internet-Draft, Jan. 2021. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-transport/

[16] J. Moy, "OSPF Version 2," IETF, RFC 2328, Apr. 1998. [Online]. Available: http://tools.ietf.org/rfc/rfc2328.txt

[17] R. Coltun *et al.*, "OSPF for IPv6," IETF, RFC 5340, Jul. 2008. [Online]. Available: http://tools.ietf.org/rfc/rfc5340.txt

[18] S. Alexander and R. Droms, "DHCP Options and BOOTP Vendor Extensions," IETF, RFC 2132, Mar. 1997. [Online]. Available: http://tools.ietf.org/rfc/rfc2132.txt

[19] V. Fuller and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan," IETF, RFC 4632, Aug. 2006. [Online]. Available: http://tools.ietf.org/rfc/rfc4632.txt