Increasing Resilience of SD-WAN by Distributing the Control Plane

Friedrich Altheide, Simon Buttgereit, Michael Rossberg, Guenter Schaefer
Technische Universität Ilmenau
firstname.lastname@tu-ilmenau.de

***Abstract*—Modern WAN interconnects utilize SD-WAN to automatically respond to network changes and improve link utilization, latency, and availability. Therefore, they incorporate some kind of centralized controller that collects network state from all managed gateways, calculates suitable forwarding actions, and distributes them accordingly. However, this limits the robustness and availability of the network control plane, especially in the event of node or partial network outages. In this paper, we propose a distributed and highly robust SD-WAN control plane without any central or regional controller. Our solution can handle arbitrary device failures as well as network partitioning. The distributed forwarding decisions are based on user-defined, dynamically evaluated path cost functions, and consider not only path quality but also quality fluctuations. The evaluation shows that our approach can handle several thousand SD-WAN gateways and multiple hundred network policies in terms of computation. Yet, it also highlights that distributing all decisions requires additional communication bandwidth, which may limit the number of supported connections in certain scenarios.**

***Index Terms*—SD-WAN, SDN, Robustness, Distributed Systems**

I. Introduction

To enable per-application-based traffic engineering, modern networks are continuously monitored to automatically react to network changes and reroute traffic, resulting in improved overall network throughput, per-flow throughput, end-to-end latency, and ultimately, quality. The presumably most common technique to achieve such an automatic, fine-grained traffic control is Software Defined Networking (SDN) and more specific, when connecting remote locations over a Wide Area Network (WAN), Software Defined WAN (SD-WAN). Essential to this approach is a centralized controller that is programmed with an abstract policy, defining how to handle traffic. Unlike a static configuration, this policy can be dynamically updated to adapt to changing requirements. The controller regularly gathers network topology information and the global network state, such as node and link utilization or link latency. With this knowledge and based on the global policies, it can determine where to route specific traffic flows. Corresponding forwarding rules are then sent to the SDN devices, which realize the forwarding decisions. This method enables the programming of complex network behaviors with relatively simple and comprehensive network programs, which are executed on the controller. Today's widely deployed SDN and SD-WAN solutions offer comparatively simple, automatic, flexible, and fine-grained traffic control to optimize routing both per network flow, but also on global scale. But to retain reactivity to link failures or overload situations, a continuous connection to (some kind of) centralized controller is required. Although it is common to deploy multiple controllers across the controlled infrastructure, they still pose a neuralgic point in the event of network partitions or limited reachability. Consequently, the robustness and availability of the network control plane are decreased significantly.

We state, a solution for an automatic, fine granular traffic control should support the implementation of global network policies on a global and local scale and facilitate:

- decisions based on the current Quality of Service (QoS) properties,
- flexible packet matching,
- symmetric traffic flows, e.g., to cope with Network Address Translation (NAT) and stateful firewalling,
- high availability gateway support, e.g., with geo-redundancy, and
- simple firewalling (blocking of flows).

Furthermore, a solution should be:

- highly available,
- robust against network failures and partitioning as well as arbitrary node failures,
- reactive towards network changes, yet provide stable end-to-end connections,
- be able to handle thousands of SD-WAN gateways,
- offer a high flexibility for an administrator, yet still be comprehensible and
- should be independent of any kind of exposed infrastructure.

Unlike existing solutions, this paper presents a SD-WAN architecture that satisfies the stated requirements, by fully distributing the control plane. By dividing global network policies into gateway-local policies and selecting WAN connections based on user-defined cost function values, as well as estimations of value fluctuations, we achieve highly robust and available traffic engineering that remains responsive even in the event of device and network failures.

The rest of the paper is structured as following: In the next section the current state of the art is presented. Section III and IV describe new methods of consequently distributing the SD-WAN control plane and how to program such a distributed system. Following, in section IV the proposed architecture is evaluated. The paper closes with a conclusion and a description of possible future work.

979-8-3503-3807-2/23/$31.00 ©2023 IEEE

II. Related Work

One solution to overcome the described issues resulting from a centralized architecture could be to use well established distributed routing protocols. The Border Gateway Protocol (BGP) is the major routing protocol for large networks. It offers exceptional scalability, availability, and robustness by avoiding the need for a central control instance. Further, BGP supports flexible parameterization to differentiate paths through a network. Yet, aside from destination prefixes, it is not capable of routing fine granular traffic flows over different paths without sacrificing scalability. Additionally, BGP does not natively support the representation of bandwidths or latency in its parameters, making it unable to automatically react to changes of those network characteristics. These limitations apply to all common routing protocols.

To support a fine-grained traffic control while overcoming the described issues of SDN, a common approach is to deploy distributed controllers. The most simple design is the usage of a clustered controller infrastructure where the state is synced between the cluster elements. Every SDN device is assigned to one controller [1]. This assignment can either be static or dynamic to support automatic load balancing between the cluster elements [2], [3]. If synchronization between the controllers is done with some kind of consent protocol such as Paxos [4] the number of cluster nodes and latency between the members, as well as the amount of consensus required [5] become a limiting factor. To limit this problem, some SDN controllers use eventual consistency [6], yet the amount of data to synchronize becomes enormous for a large number of SDN controllers. While horizontal clustering improves the availability of the cluster itself, the reactivity of the controlled network remains limited when gateways cannot reach the controllers due to a network partition. Hence, clusters are typically distributed among different regions to compensate regional network outages [7], [8]. While this improves the availability, it also introduces additional latency to the synchronization messages, thus limits the scalability.

This limitation can be lifted by using a hierarchical controller design [9]–[11], which splits the network in several independent network regions. Each region is managed by its own independent regional controller. To steer traffic between regions, a global controller is introduced. Hence, failures of a regional controller do not affect other regions. Although this design seems promising and offers good scalability, it suffers from availability issues as the routing between regions cannot adapt to network changes, e.g., high link utilization and changed error rates, once the global controller becomes unreachable. Hence, partitions between controllers can propagate to the inter-regional traffic even if the network between the controlled gateways is not partitioned.

III. Distributing the SD-WAN control plane

In this work, we propose to distribute the control plane and thus the decision process into the distributed SD-WAN gateways (see Fig. 1). These distributed entities must then coordinate how to route flows while achieving scalability and minimizing the control overhead. The entire concept is designed to function without the need for an external instance. It ensures that any pair of sites, which is able to communicate, directly coordinates their respective intra-site traffic. Hence, a failure of one site or its gateways only affects connections within or with that site, while other sites are unaffected and can still realize their network policies. But, to achieve high availability (A) and partition tolerance (P), it may be necessary to sacrifice strong consistency (C) as explained by the CAP theorem [12]. Consequently, protocols such as Paxos [4], are not suitable. Instead, we employ the method of eventual consistency [13], which ensures that the entire network eventually reaches a consensus on how to handle network traffic. This means that, after a network failure, any discrepancies or conflicts that arose between the gateways will be resolved over time, and the network will operate uniformly.

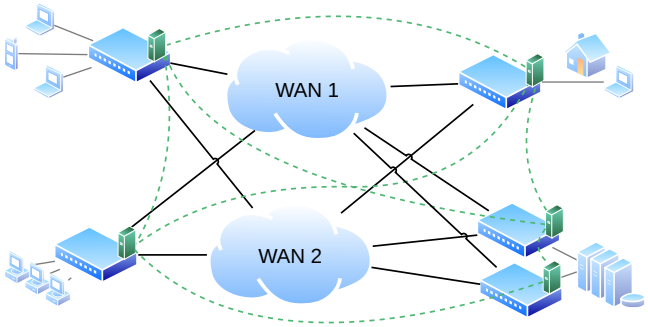

Fig. 1. SD-WAN controller elements (green) are placed on each gateway and coordinate their decisions on whether and how to route/distribute network traffic between their sites.

A. Distributed global network policies

A global SD-WAN policy typically comprises two parts: a traffic description and a requirement on how to handle the traffic. To implement such a policy, a SD-WAN gateway must first determine whether its site is relevant for the realization of the policy. If it is, and the policy involves connectivity between multiple sites, the gateway must search for a path that satisfies the policy requirement and install appropriate forwarding rules into its data plane. If the policy only involves traffic within the site, it is considered a local SDN policy and is outside the scope of this work.

Coordinating decisions among a potentially large number of gateways can be complex and may not scale well if done improperly. In existing SD-WAN solutions, these steps are typically handled by central SD-WAN controllers that have global knowledge of the network topology and state. However, this approach has robustness issues, e.g., prevents partition tolerance. While a centralized instance may be necessary for some complex SD-WAN policies to coordinate all decisions, it's typically not required for most. This is because SD-WAN policies usually involve paths between at most two sites, which means that coordination on routing decisions can be limited to those two sites alone.

Global policy:

For all kind of traffic use low-latency connections

Inter site policies:

1. For traffic between site A and B use low-latency connection.
2. For traffic between site A and C use low-latency connection.
3. For traffic between site B and C use low-latency connection.

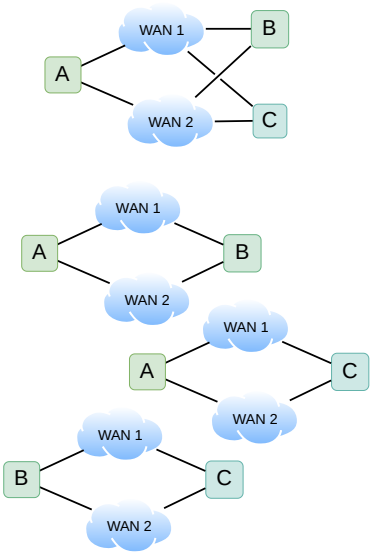

Fig. 2. Global policy is transformed into three inter-site policies between each gateway pair.

To describe policies that are limited to one pair of sites we introduce the term *inter-site policy*. A global network policy is transformed into multiple inter-site policies, each describing a flow between exactly two sites (see Fig. 2). In doing so, a local decision-making by SD-WAN gateways becomes feasible (details follow in Sec. III-B).

The transformation of a global policy into multiple inter-site policies can be executed by a centralized management/-monitoring component, a local administration tool, or even on the gateway itself. Note that while an external transforming component may be necessary during the initial phase, once the inter-site policies are appropriately distributed, it does not affect the availability or robustness of the decision algorithm itself and therefore represents no issue for the overall robustness.

B. Inter-Site policy realization

We aim to support two types of SD-WAN policies: Firewalling (drop) policies and routing policies. Firewalling policies do not require advanced synchronization because each gateway involved can independently check packets to determine whether to forward or drop them. Routing policies, in contrast, require coordination between the participating instances. To enable stateful firewalling and NAT, it is necessary for flows between two sites to use symmetric paths for both directions. Therefore, protocols that allow each site to make independent decisions should be avoided. Instead, both sites must coordinate their decision of which WAN connection to choose. In the context of this paper, a connection refers to an end-to-end association between two SD-WAN gateways, which can be established through a variety of means such as a VPN tunnel, an MPLS label path, a VLAN, or simply a specific QoS tag. As the available connections between two sites may have a high latency, e.g., due to the physical distance, classical consensus

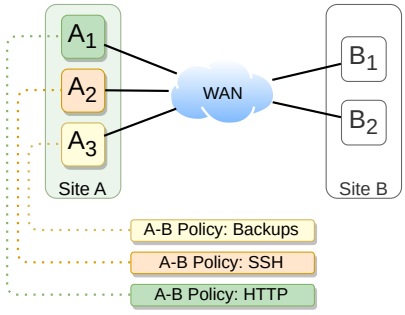

Fig. 3. Two multi-gateway sites A and B are configured with three inter-site policies with site A being the leading site. To achieve a good scalability, the role of the inter-site policy leader for the three different policies can be distributed across gateway A_1, A_2 und A_3.

protocols may limit the responsiveness. Instead, we propose to deterministically select one site as the leading site. The leading site is then responsible of choosing a single connection for each inter-site policy between itself and the remote site. The latter has to follow these decisions. To select the leading site for each pair of sites, we compare the sites' ID concatenated with the remote sites' ID and prefer the smaller hash:

$$hash(ID_{\text{site}_1}|ID_{\text{site}_2}) \overset{?}{<} hash(ID_{\text{site}_2}|ID_{\text{site}_1})$$

This comparison prevents the site with the smallest *ID* from always taking the role of the leading site. Instead, the number of leader roles per site is evenly distributed in larger networks (details follow in evaluation).

For reasons of high availability, one or both sites may consist of multiple SD-WAN gateways. To allow a good scalability, availability as well as robustness, we propose to further distribute the decision-making of the individual inter-site policies within the leading site.

We therefore introduce the role of *inter-site policy leaders*. Gateways assigned to this role make all decisions for a concrete inter-site policy. Other gateways within the same site, as well as those in the remote site, then follow the decisions made by the leader for that particular inter-site policy. Since a site typically will be configured with multiple inter-site policies, the different inter-site policy leaders can be distributed across all SD-WAN gateways of the leading site (see Fig. 3).

The decision-making is illustrated for three sites as an example in Fig. 4. It starts with each gateway notifying the inter-site policy leader of all connections that are appropriate for the given policy by transmitting the current network quality (more details follow in Sec. IV). This action is triggered by a local and policy-specific *statistic timer*. When a *decision timer* triggers, the leader gateway selects a WAN connection/QoS tag etc. and disseminates the decision among all gateways of both sites. If a new decision is made that switches to a different connection, the SD-WAN gateways generate and install new forwarding rules into their data plane. In case a remote gateway is not directly reachable by the leader gateway, but via another gateway, the latter relays information and decisions. These steps are repeated periodically and can be

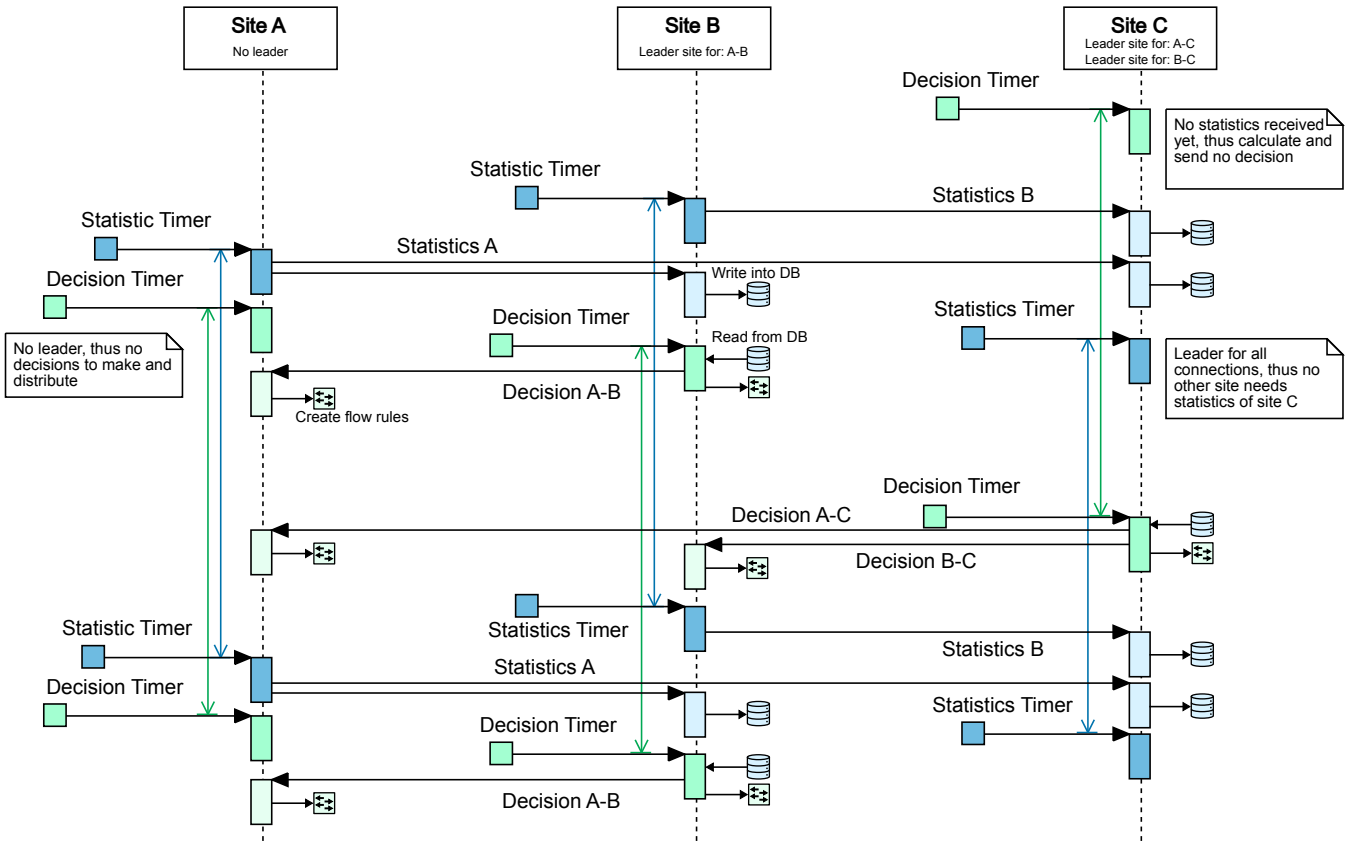

Fig. 4. Time sequence of decision making between three sites. Site A with no leader role only sends statistics, receives decisions and writes appropriate flow rules into the data plane. If a site has one (B) or multiple leading roles (C), it receives and stores statistics. Based on these, a decision is calculated, and then transmitted to the appropriate remote site. After a decision was received by a site, flow rules are written into the data plane of the sites gateways. Both steps of sending statistics and calculating decisions are triggered by timers. For sake of readability, the statistic timers as well as the decision timers of one site are executed at the same time. It should be noted that in reality all timers operate independently and are not synchronized in any way. The process of sending/receiving statistics and calculating/receiving a decision for a specific inter-site policy can be divided into rounds. Each round lasts for one timer interval and starts at the moment when the statistic timer is triggered.

divided into rounds/intervals. It should be noted that each gateway can take on the leader role for multiple inter-site policies simultaneously, e.g., site C in Fig. 4 for *A-C* and *B-C*.

To limit the impact of gateway failures, the inter-site policy leader role is assigned to the gateway through which the policy's network flow is currently sent. This approach enhances both robustness and availability compared to randomly distributing inter-site policy leader roles. After startup, each gateway first selects itself as the leader for all inter-site policies where it is part of the leading site. If a gateway discovers another gateway within the same site that it is currently unaware of, it establishes a bidirectional channel. Both gateways then exchange information about the inter-site policies they are currently leading and the quality of their selected connections. In cases where both gateways found no connection or their connections have equal quality for a particular policy, the gateway with the higher *ID* relinquishes its role, resulting in the establishment of a single leader. In all other cases the gateway with the better connection wins and stays the policy leader. After both gateways exchanged their leading roles, exactly one leader exists for every inter-site policy. If the leader selects to route the flow via another gateway, it informs all gateways and hands over the policy leader role.

To remain responsive in the event of a leading gateway failure, decisions have a lifetime $t_{exp} = c \cdot t_{timer}$ with $c > 1$. Each gateway in the leading site periodically checks for expired decisions. For each expired decision, the gateway announces itself as the new leader to all other gateways within the site. The presented rules ensure that over time only one inter-site policy leader will remain, even if multiple gateways choose to promote themselves as leader simultaneously. Even if the gateways of a site currently do not reach each other, the protocol ensures that eventually one leader is chosen and a symmetric decision can be made for each inter-site policy independently.

IV. Programming a Distributed Control Plane

By utilizing the described architecture, decisions can be made in a fully distributed manner, eliminating the requirement for a central component or global/regional knowledge. Thus, telling the control plane on how to react to changing network

```
fn costs(con_stats, util, time_stamp) -> f64 {
  if(time_stamp between 8am and 10am) {
    // classify connections in 10ms steps
    return con_stats.latency().as_ms().round() / 10
  }
  if(time_stamp between 10pm and 4am) {
    // select no connection and block traffic
    return f64::MAX;
  }
  // prefer connection with lowest latency
  return con_stats.latency().as_ms();
}
```

Listing 1. Based on the connection statistics and the current time a programmed metric function returns a weight.

characteristics, is not realizable by one simple network program deployed on a central component. Yet, for reasons of usability, a simple but comprehensive method of programming the distributed system is required. We therefore propose to let the distributed gateways weight every connection, based on locally available data such as connection statistics, gateway utilization and/or the current time. The calculation of a specific connection's weight is accomplished through the use of customizable metric functions within the SD-WAN gateways. These functions, as demonstrated in Listing 1, determine the weight > 0 for each connection. The idea is to let the inter-site policy leader compare different connections using their calculated weights with lower cost considered as the better connection. As a connection involves two endpoints, either both endpoints weight their statistics independently and send the weight to their policy leaders, or the connection statistics are transferred to the leading site, where they are weighted and then sent to the leaders. In both cases the leader can then sum both weights to a connection weight which is then used for comparisons. If the goal is balancing the execution time between both sites, it is beneficial to evaluate the statistics independently on each endpoint. Yet, the weights have to be transmitted per metric and connection, which results in a linear increase of the required bandwidth to exchange the weights. Hence, sending weights is only suitable for networks with only few metric functions. For networks with a high number of different metrics, statistics should be transmitted to the leading site instead. This approach can significantly reduce the required bandwidth as statistics are only sent once per connection. The statistics are then weighted by the gateways at the leading site and transmitted to the corresponding inter-site policy leaders. Details on this can be found in the evaluation section V-C.

Either way, the programmed metric functions used to weight the statistics should be easily changeable to provide high flexibility for administrators. Otherwise, it would not be possible to adapt existing steering properties or introduce new QoS requirements. Further, robustness (e.g., limiting resource consumption) and security must be ensured, e.g., by isolating the metric function execution. To fulfill these requirements, we utilize the binary format WebAssembly (Wasm) [14], which not only is executed with near native speed inside web browsers, but also in isolated execution environments called WebAssembly System Interface (WASI)[1]. It allows a user to write `costs()` functions in any supported language[2], compile them into a textual representation and append it to the global and subsequently to the inter-site policies. The binary is then loaded into the gateway's WASI execution environment and used to calculcated the weight of a connection for a specific policy.

A. Making stable decisions

After receiving the weights of all connections, each inter-site policy leader needs to select one distinct connection for their policy. Like any traffic control algorithm, we aim to increase the quality of all flows, while also ensuring stable end-to-end connections. Yet, optimizing the quality of inter-site policies presents several challenges that the selection algorithm must address:

1) Selecting the best available connection for all inter-site policies might decrease the quality of that connection.
2) Load balancing different inter-site policies onto the n metrical best connections might decrease quality, if n is chosen too small (e.g., bandwidth exhaustion) yet also, if chosen too high, e.g., higher delay for flows routed over n^{th} best connection.
3) Switching traffic of multiple inter-site polices simultaneously might (negatively) affect each other.
4) Inter-site policies might continuously toggle between connections, since switching could decrease the quality of the selected connection while the previous connections quality might recover, due to reduced utilization.
5) If one connection consistently offers slightly better costs over an extended period of time for an inter-site policy, it should be used to improve overall quality. The connection switch should neither be too fast (e.g., only temporary low costs) nor too slow (poor quality until switch is performed).

These challenges do not only occur because of dynamically programmed metric functions, but also because of varying WAN quality, as well as traffic flows in number and size. Furthermore, objectives can conflict with each other, like reducing toggles between two connections and preferring the better connection in the long run. As a result, relying on a static threshold per metric and switching when it is exceeded will not adequately address the challenges described. Instead, a dynamic threshold is necessary. Its evaluation has to be efficient in computation but also in memory consumption. Hence, incremental calculations are preferred.

As a solution for the presented challenges, we propose a *class of best connections*. This class includes all connections whose current metric falls within the fluctuation range of the best available connection $f(con_{best})$ (see Fig. 5). This fluctuation could be calculated by first smoothing out outliers, e.g., with a moving average, and then determining the deviation

[1]https://wasi.dev/

[2]https://github.com/appcypher/awesome-wasm-langs

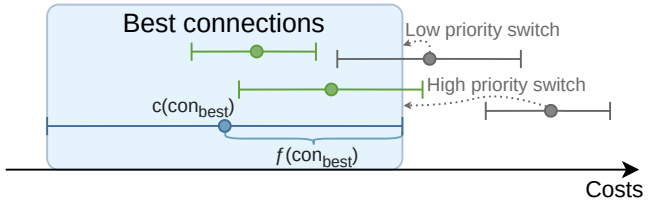

Fig. 5. All connections with costs inside the fluctuation of the "best" connection $c(con_{best}) \pm f(con_{best})$ are included into the best connection class. If the currently used connection is outside this class, the inter-site policy should be switched to one of the class members (dotted lines). Depending on the current cost difference, the switch is scheduled soon or later.

from the current value (which could also be smoothed out by a moving average). When applying this method first for testing, we observed that the fluctuation of the best connection quickly converges to a value close to 0 if the metric weight is nearly static, e.g., the delay on a low utilized link. Yet, in case of an extremely short quality degradation, the fluctuation also stays close to 0. Thus, our approach to calculate the fluctuation does not seem to be the perfect choice to model fluctuation for all scenarios. Other methods, of limiting the attenuation by incorporating rapidly fluctuating network situations more quickly than stable situations, may be more suitable. The evaluation of different fluctuation metrics and their application in different scenarios is, however, left for future work.

Switching an inter-site policy to another connection within the *class of best connections* does not offer a statistically relevant advantage, since all metrics within the class are insignificantly different from the best connection for the given metric. Yet, if the used connection is not part of the class, a (random) connection within the class (see Fig. 5) should be selected to transfer the corresponding traffic over a path with higher quality. To prevent instabilities due to many switches at the same time, the moment of switching is randomly chosen using a Bernoulli distribution. The probability p whether to switch to another connection in this decision interval/round is determined by the difference between the current metric value $c(con_{cur})$ and the upper limit of the *class of best connections* $c(con_{best}) + f(con_{best})$.

$$p = \min\left(\alpha, 1 - \frac{c(con_{best}) + f(con_{best})}{c(con_{current})}\right) : \alpha \in (0, 1]$$

If the previous connection is only slightly better than the best connection, p will diverge to the minimal switching probability α. Otherwise, the value of p will be between α and 1, depending on the difference between the connections. Therefore, α influences how long a inter-site policy stays on a connection that is nearly similar to the best connection but not part of the *class of best connections*. In Sec. V-A, we evaluate the duration required for a connection switch, taking into account different values of p.

V. Evaluation

The evaluation of a distributed SD-WAN solution is oriented on the requirements, stated in section I. Support for flexible, QoS-based routing and firewalling decisions as well as symmetric routing follows directly from the architecture, described in the previous two sections.

In the following, we first evaluate the time required to respond to changing network conditions. We will then discuss robustness and support of high availability, especially in case of geo-redundant gateways, and its scalability. The scalability evaluation is partially based on an initial prototype that we had built for conducting feasibility tests. Yet, some initial assumptions led to a varying scalability of different factors (details follow in Sec. V-C). Thus, we leverage the linearity of the different factors to extrapolate the behavior for high numbers of gateways, connections, and policies. Although this method may not be the most precise way to measure scalability, it provides an insight into the concept's capabilities as well as potential challenges when implementing it.

A. Reaction time

To evaluate the reaction time, we will first discuss how fast our approach can react to network changes such as failing connection. Next, we will calculate the duration for which a connection switch can be artificially degraded to enhance the overall network stability, if the currently selected connection is only marginally outside the *class of best connection*.

1) Minimal reaction time: The routing decision for a specific policy involves several communication steps that impact the minimum reaction time. Initially, an inter-site policy leader must be selected. Next, the network conditions are regularly sent to the leader. The leader then calculates the appropriate decision and distributes it. In the event of a network or device failure, the failure must first be detected, and a new leader may need to be determined. The process of recalculating and distributing a decision is then repeated.

The selection of the leading site is done implicitly when setting up a connection. The selection of the correct policy leader then takes one message inside the cluster and happens, when a policy is inserted. Both steps can be considered as setup and do not add to the reaction time required for regular network condition changes.

Each gateway regularly collects local information and transmits them (statistics or precalculated metrics) every t_{timer} to the leader. Updates from a remote site take $1/2$ RTT$_{inter}$ before beeing received at the leading site. Data measured at the leading site takes $1/2$ RTT$_{intra}$ to reach the inter-site policy leader. Every t_{timer}, the policy leader calculates a decision based on the received data and sends it to all gateways participating in the policy. If a remote gateway has no direct connection to the leader, updates to and decisions from the leader have to be forwarded by other gateways of the leading site adding $1/2$ RTT$_{intra}$ per direction. All in all, the reaction time sums up to:

$$t_{react} \leq \text{RTT}_{intra} + \text{RTT}_{inter} + 2 \cdot t_{timer}$$

In local clusters, members are expected to communicate without significant delay resulting in RTT$_{intra} \approx 0$. Futher, both the statistics and decision timer, have a expected delay of

$1/2\ t_{timer}$ until the next trigger. Consequently, the expected reaction time is:

$$E(t_{react}) = \text{RTT}_{inter} + t_{timer}$$

The gateway that realizes the inter-site policy's routing automatically becomes its leader. Hence, a failure of the currently used connection can be handled immediately, by calculating an appropriate response and informing both the local and remote gateways:

$$t_{con\_failure} = \frac{1}{2}\,\text{RTT}_{intra} + \frac{1}{2}\,\text{RTT}_{inter}$$

Gateways at the leading site detect the failure of a policy leader when decisions are not renewed within its lifespan t_{exp}. If a gateway detects a failure, it instantly assumes itself as the new leader, calculates a new decision, and distributes it. As multiple gateways may have elected themselves, a uniform routing decision is reached after all cluster members distributed their decisions and a single leader is chosen using the rules described in Sec. III-B. Thus, the reaction time after a failure of the leading gateway is:

$$t_{lead\_failure} < t_{exp} + \frac{1}{2}\,\text{RTT}_{intra} + \frac{1}{2}\,\text{RTT}_{inter}$$

It should be noted, that, if the Transmission Control Protocol (TCP) is used for gateway communication one additional Round Trip Time (RTT) has to be accounted for the initial handshake. Yet, this is typically performed, when initializing connections, thus does not add onto the reaction time.

2) Artificial reaction time degradation: To maintain a high robustness and not immediately switch a connection to a supposedly better one, Sec. IV-A introduced the concept of artificially extending the reaction time. Yet, even if a connection switch is delayed, eventually all inter-site policies should use a connection inside the class of best connections. Fig. 6 displays the probability of switching a connection when n decision intervals (rounds) have passed. The different values of p describes the difference between the current metric and the *class of best connections*.

If the decision interval is configured to be 10 seconds, inter-site policies where the metric is comparatively poor ($p = 0.7$) have a $99.9\,\%$ chance for switching into the *class of best connections* in under 1 minute/6 rounds. For policies with quite similar metric values ($p = 0.1$) it takes 11 minutes/66 rounds to reach the same probability. Thus, a connection, which briefly leaves the class of best connections, will not lose the policies assigned to it. Nonetheless, every policy whose metric value is not inside the *class of best connections* for a long time will eventually switch into it. It should be noted, that the artificial reaction time degradation is not applied to switches after connection failures, where the fastest possible switch is always used.

B. Robustness and High Availability Support

Sec. III describes how to fulfill the stated functional SD-WAN requirements without requiring any kind of exposed infrastructure component. By using inter-site policies instead of global

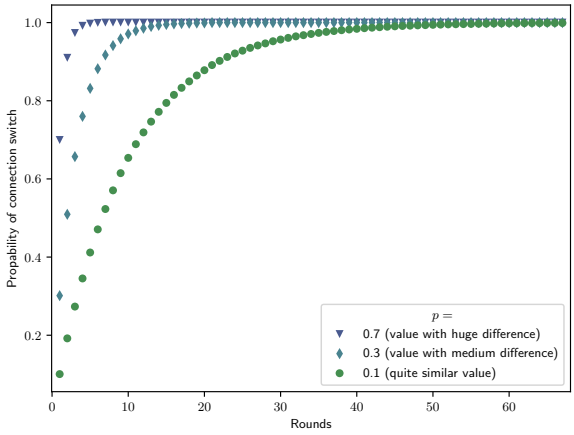

Fig. 6. Probability of switching into the class of best connections for different values of p.

policies and selecting an inter-site policy-specific leader, only one instance controls the routing of a specific traffic flow. Thus, if two gateways are able to communicate, they are also able to coordinate decisions to realize the optimal path according to SD-WAN policies. The proposed algorithms in Sec. III-B ensure that independent from any previous network failure scenario eventually ($1/2\,\text{RTT}$ after connectivity is regained) a single leader persists per policy. The algorithms not only support a connection between single-gateway sites but also gateway clusters. If a gateway fails, all its decisions time out, and the appropriate inter-site policy leader roles are taken over by another gateway of the leading cluster. The described algorithm ensures that only one gateway persists as leader for a specific policy.

To ensure high robustness and availability, even in the case of geo-redundant clusters, the architecture must be capable of handling high-latency intra-cluster connections. Sec. V-A shows that the reaction time on changing network conditions and failures depends on the RTT_{intra}. Yet, the impact is only additive and does not depend on the number of cluster gateways. In contrast to the local cluster, the delay between members of a geo-redundant clusters is subject to its physical distance. Hence, RTT_{intra} can be expected similar to RTT_{inter} resulting in:

$$E(t_{react\_geo}) = 2 \cdot \text{RTT}_{inter} + t_{timer}$$

The formulas for calculating the reaction time after network and gateway failures remain the same between geo-redundant and non-geo-redundant scenarios. In case, two gateways of a (geo-redundant) cluster do not reach each other because of a network failure, the cluster falls into two partitions. Nodes that do not reach their current policy leader start the process of determining a new policy leader using rules stated in Sec. III-B. Thus, a second cluster, which acts independently from the first one, forms. After communication between both partitions is regained and the connections are reestablished the

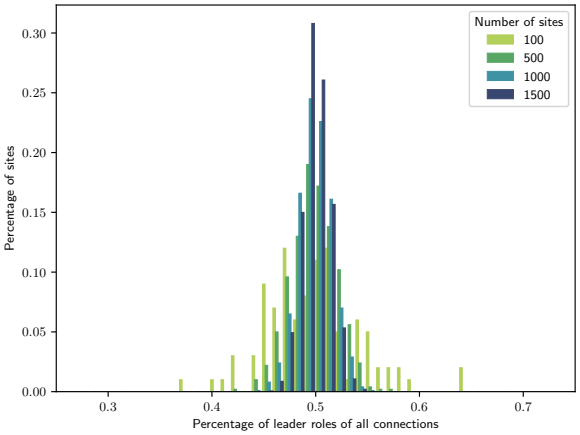

Fig. 7. Distribution of percentage of leader roles between varying number of sites with a standard library *SipHash* algorithm.

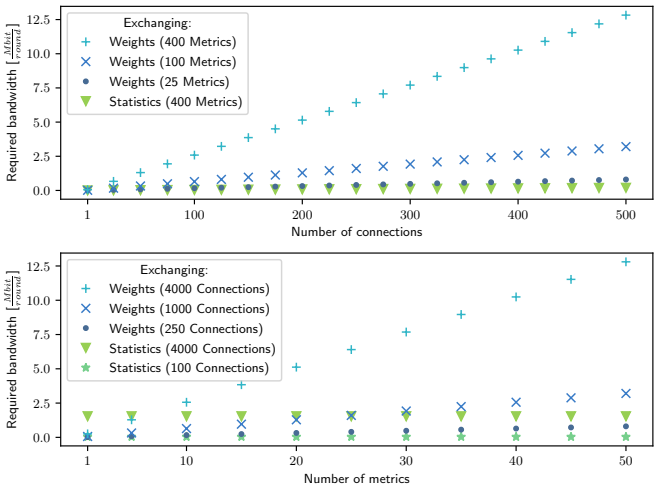

Fig. 8. Extrapolated bandwidth requirements (at one site) for exchanging metric costs or statistics. For metric numbers < 10 sending precalculated metrics requires less bandwidth. Yet, in every other situation sending raw traffic statistics requires significant less bandwidth.

initial policy leader algorithm is executed again. Subsequently only one inter-site policy leader remains.

C. Scalability

In Sec. V-A, we observed that the reaction time of the concepts is not a critical factor for scalability. This is because it remains constant regardless of the number of sites, gateways, or gateways per site. Instead, the scalability of the overall solution is primarily determined by three key factors:

1) The execution time and memory usage of the programmable metric functions.
2) How evenly the evaluation tasks can be distributed among multiple gateways.
3) The amount of data that needs to be transferred between gateways and a leader.

When building the prototype, we discovered that the first two factors pose not such a big challenge as factor three.

1) CPU and memory usage: The CPU usage scales linear with the number of metric calculations. This corresponds to the number of inter-site policies $\times$ the number of connections. In the following, this product is called *policy connections*. One policy connection represents one inter-site policy realized for one connection to choose from. In our prototype, a simple metric function requires $775\,\mathrm{kB}$ of memory and takes approximately $3.4\,\mu\mathrm{s}$ to calculate on one $2.20\,\mathrm{GHz}$ (Xeon Gold 5120) core.

As an example, if one CPU is reserved for the distributed SD-WAN software, configured with a $10\,\mathrm{s}$ interval time, a gateway could handle up to $\sim$2 800 000 policy connections (including $\sim 2.0\,\%$ base utilization). Breaking down the 2 800 000 policy connections into 2000 sites connected over two WANs, a gateway would be configured with 700 different policies. Assuming every policy has its own metric function, about $550\,\mathrm{MB}$ of storage would be needed for the metric function handling.

2) Distributing the calculations: Distributing the leader roles equally among all sites reduces the number of calculations required for an individual gateway by half. Fig. 7 shows that by utilizing the deterministic selection of a leader site (as described in Sec. III-B) along with a standard library hash algorithm (*SipHash* [15]), an equal distribution can be achieved if the number of gateways is sufficiently high.

Additionally, if the calculations are distributed among multiple gateways, e.g., inside a cluster, this number can be increased furthermore. Yet, the intra-cluster distribution depends on the realization of the concrete inter-site policies. One well connected gateway might handle all policies and become leader for all policies. Thus, for cases where all metric functions are quite similar or one cluster member offers a comparably good connection, appropriate computing resources should be provisioned.

In most cases, distributing calculations over multiple gateways will not reduce memory usage because the metric functions must be stored on all gateways to enable rapid response to a policy leader change.

3) Bandwidth overhead: Due to our fully distributed architecture, control data is exchanged between all connected gateways instead of sending it directly to a single data center. This leads to a maximum of $n-1$ instead of 1 communication partner per gateway. If a limited number of metrics is used, sending the precalulated metric values (8 bytes per metric and connections) is more efficient then sending raw data plane statistics (48 bytes per connection, both depicted in Fig. 8). Yet, for high numbers of different metric functions bandwidth requirements vastly exceed acceptable limits. Thus, our prototype should be sending raw traffic statistics instead of the precalculated metric values to achieve a good scalability in relastic scenarios. Nevertheless, it should be noted that an overhead of $0.8\,\mathrm{Mbit/round}$ (sending statistics for 2000 con-

nections) might become significant for a low bandwidth site, e.g., with a DSL connection. To further reduce the required bandwidth, subscribing only to the necessary data and sending only incremental changes could be further evaluated. Note that if metric functions are enriched by e.g., device specific statistics like uplink utilization and CPU usage, slightly more precalculated metric values can be exchange before reaching the point of break even for sending statistics. Therefore, the prefered approach remains to exchange statistics.

VI. Conclusion & Future Work

This paper presented a novel approach that fully distributes the control plane of SD-WAN, which enables automatic, fine-grained traffic control without introducing exposed entities. As a result, the network control plane remains highly available and robust, even in the event of device failures or network partitioning. This is achieved by three methods. The first is, to split global network policies into local inter-site policies. This eliminates the need for complex consent algorithms when deciding how to route traffic and enables a distributed decision over policy based routes, even if gateways are placed in a geo-redundant clustered setup with high communication latency. Second, to enable a simple and comprehensible way of programming the distributed control plane, we propose to attach programmable metric functions to the policies. These functions are executed inside the distributed instances to evaluate the quality of appropriate connections. Third, the paper introduced a method for dynamically selecting an optimal path for routing without introducing instabilities into the network. This is achieved by considering not only the current quality of a connection, but also its fluctuations. The quantitative evaluation shows that this solution is capable of managing several thousand nodes and inter-site policies in terms of computation. However, by avoiding a central controller, statistics and decisions have to be directly exchanged between the gateways. Hence, more control connections have to be maintained and device-specific statistics are transmitted multiple times instead of sending them once. In future work, we will assess the impact of different approaches to reduce control traffic, such as sending only incremental changes, on the reactivity of the distributed SD-WAN control plane. Further, we plan to evaluate the scalability and robustness in large real world setups.

It should be noted that the described methods may not only be used in combination, but could also be applied separately. Considering the fluctuation of an input value instead of using only thresholds adds network stability to any SDN/SD-WAN controlled network. To go further, distributed and dynamically programmable metric functions could be applied to any distributed optimization tasks, such as packet routing, network function placement or resource dimensioning in ad hoc networks. In summary, this paper introduced new methods for achieving a highly robust and available traffic engineering solution that can implement fine-grained global policies while remaining responsive even in the event of device failures, network failures, or network partitioning.

References

[1] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," *CoRR*, vol. abs/1401.7651, 2014. [Online]. Available: http://arxiv.org/abs/1401.7651

[2] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2014, pp. 17–27.

[3] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 18–25.

[4] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, p. 133–169, may 1998.

[5] T. Zhang, P. Giaccone, A. Bianco, and S. De Domenico, "The role of the inter-controller consensus in the placement of distributed SDN controllers," *Computer Communications*, vol. 113, pp. 1–13, 2017.

[6] E. Sakic and W. Kellerer, "Impact of adaptive consistency on distributed SDN applications: An empirical study," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2702–2715, 2018.

[7] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10. USA: USENIX Association, 2010, p. 3.

[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USA: USENIX Association, 2010, p. 351–364.

[9] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 19–24.

[10] Y. Fu, J. Bi, K. Gao, Z. Chen, J. Wu, and B. Hao, "Orion: A hybrid hierarchical control plane of software-defined networking for large-scale networks," in *2014 IEEE 22nd International Conference on Network Protocols*, 2014, pp. 569–576.

[11] D. Marconett and S. B. Yoo, "Flowbroker: A software-defined network controller architecture for multi-domain brokering and reputation," *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 328–359, 2015.

[12] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[13] P. Bailis and A. Ghodsi, "Eventual consistency today: Limitations, extensions, and beyond," *Communications of the ACM*, vol. 56, no. 5, pp. 55–63, 2013.

[14] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017, pp. 185–200.

[15] J.-P. Aumasson and D. J. Bernstein, "SipHash: a fast short-input PRF," in *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*. Springer, 2012, pp. 489–508.