Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

sRetor: a semi-centralized regular topology routing scheme for data center networking

Zequn Jia[1,2], Qiang Liu[1*] and Yantao Sun[1]

Abstract

The performance of the data center network is critical for lowering costs and increasing efficiency. The software-defined networks (SDN) technique has been adopted in data center networks due to the recent emergence of advanced network control and flexibility demand. However, the rapid growth of data centers increases the complexity of control and management processes. With the rapid adoption of SDN, the following critical challenges arise in large-scale data center networks: 1) extra packet delay on the separated control plane and 2) controller bottleneck in large-scale topology.

We propose sRetor in this paper, a topology-description-language-based routing approach for regular data center networks that leverages data center networks' regularity. sRetor aims to reduce the packet waiting time and controller workload in software-defined data center networking. We propose to move partial forwarding decision-making from the controller to switches to eliminate unnecessary control plane delay and reduce controller workload. Therefore the sRetor controller is only responsible for troubleshooting complicated failures and on-demand traffic scheduling. Our numerical and experimental results show that sRetor reduces the flow start time by over 68% and the failover time by over 84%.

Keywords Data center networking, Regular network topologies, Software-defined networking, Topology description language

Introduction

With the development of technologies such as cloud computing [1, 2], virtualization [3] and 5G/6G communication [4–6], the scale effect of data centers is attracting the attention of both academia and industry. Various large corporations, such as Google and Microsoft, are building their own data centers by reducing the operation cost of their information systems, and the scale of their data centers is constantly expanding [7]. However, as one of the critical components of data centers, the network gradually becomes a bottleneck limiting the growth of the data center. Traditional link-state routing protocols such as OSPF are widely used, yet they generate heavy routing message overhead and consume long convergence time in large-scale data center networks [8].

To improve the efficiency of data center networks, researchers have conducted studies on topology structures and routing methods for data center networks, such as Fat-Tree [9], DCell [10] and BCube [11]. Many of these routing methods are topology-aware routing methods, i.e., specifically designed for the corresponding network topology and optimized according to the topology characteristics. As for Fat-Tree, the authors designed a two-level routing table and the corresponding routing methods to generate different routing tables according to the different roles of switches (core switches, edge switches, etc.), thus achieving efficient and scalable routing methods. Guo, et al. [11] designed the BCube Source

*Correspondence:
Qiang Liu
liuq@bjtu.edu.cn
[1] School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
[2] School of Electrical and Data Engineering, University of Technology Sydney, Sydney, Australia

Routing algorithm to perform an efficient path selection by leveraging BCube's topological property of hierarchical structure and connection features. In addition to Fat-tree [9] and BCube, other network topologies have been proposed in recent years, such as LaScaDa [12] , BCDC [13] and more in [14–18].

Although these emerging network topology structures and the corresponding routing methods provide high forwarding efficiency for data center networks, these algorithms are incompatible with each other, therefore implementing these topologies and integrating them into a data center network is complicated and costly. Thus a generic topology-aware routing algorithm that can handle a wide range of data center network topologies is critical [19].

The advent of software-defined networking has enabled addressing the requirements of contemporary data center networks [20]. SDN is able to provide a more flexible and programmable networking environment [21]. Many previous works [22–29] have demonstrated the potential of SDN in harmonizing various routing methods and integration in data center networks. For instance, Portland [22] employs a scalable, fault-tolerant layer 2 data center network fabric that leverages SDN for better control and management. Similarly, Hedera [23] introduces dynamic flow scheduling in data center networks, which is made possible through the centralized control provided by SDN. Moreover, stateless flow-zone switching has been proposed to achieve reliable and lightweight source routing in data center networks, again facilitated by SDN [27].

Even though these works have made significant contributions, they focus on specific aspects of DCN management and do not fully exploit the potential of SDN in the context of topology-aware routing across a wide range of DCN topologies. In our previous work [30], we introduced controller-side Regular Topology Routing (cRetor), a routing method designed for regular data center network topologies that capitalizes on the capabilities of software-defined networking. Central to cRetor is the domain-specific Topology Description Language (TPDL), which is instrumental in defining node properties and connection relationships in regular toopologies. Furthermore, cRetor incorporates an efficient routing algorithm based on the A-Star algorithm [31] in the SDN controller, which integrates the static topology represented in TPDL with the dynamic programming capabilities enabled by SDN.

The TPDL serves as a cornerstone of cRetor. It succinctly delineates the architecture of regular topologies by categorizing nodes based on their attributes such as location and functionality. TPDL provides network devices with a basic perspective of the network topology, encompassing both nodes and connections, while also demonstrating considerable scalability. In addition, it puts forth the innovative concept of a distance formula, which explicitly articulates the mathematical relationships governing distances between nodes. This allows routing algorithms to efficiently ascertain inter-nodal distances with reduced overhead. By streamlining this foundational computation, TPDL enhances routing efficiency.

While offering centralized, dynamic management of network devices and flow scheduling, cRetor faces challenges inherent to SDN. The overhead of OpenFlow communications between switches and controllers grows rapidly as networks expand. Although individual switches generate minimal OpenFlow traffic, cumulative overhead across potentially hundreds of thousands of switches in large-scale data centers can strain controllers. This problem is compounded by the fact that controller processing capacity often bottlenecks SDN at scale [32]. Moreover, despite cRetor's ingenious replacement of LLDP discovery with TPDL-based topology management, its reliance on OpenFlow's Packet-In mechanism for initializing flow paths remains. Thus, controllers still must process Packet-In messages for each new flow, risking overload as flow quantities surge. This on-demand computation also prolongs first-packet latency for flows, potentially violating the ultra-low latency demands of time-sensitive applications.

Multi-controller solutions are frequently utilized in typical SDN networks to tackle the scalability challenge [33–35]. However, multiple controllers greatly increase the complexity of the network and introduce numerous new obstacles to SDN management and scheduling [36]. For example, multi-controller solutions often mean that optimization problems such as data synchronization, load balancing and switch assignment between controllers need to be handled. In these optimization problems, an optimal placement may not be possible, therefore careful planning is required to identify an appropriate trade-off among the metrics. As a result, these problems are rarely handled optimally at a reasonable cost [37]. Unlike them, we aim to handle the controller bottleneck problem in a novel approach on the basis of cRetor.

This paper presents an enhanced version of cRetor, sRetor (semi-centralized Regular Topology Routing), which is a semi-centralized routing scheme for data center networks. The key difference between sRetor and cRetor is that in cRetor, TPDL is only applied to the controller while in sRetor it is applied to both the controller and switches. This allows the switches to be equipped with the topology information of the entire network as well as the ability to instantly determine the distance between any two nodes using the TPDL's distance formula locally. The sRetor switches will fetch the TPDL file at the startup stage, and after initial setup, the switches

will be able to run independently. Since the basic structure of the data center networks will not change, there is no need to update the TPDL file.

Unlike typical SDNs where the control plane is entirely centralized on the controller, some fundamental control plane tasks are distributed on switches in sRetor. Without the need to consult the controller, the fundamental forwarding function can be achieved in switches using TPDL. The switches in sRetor are similar to a standard OpenFlow switch as they can interact with the controller through the OpenFlow protocol and receive flow table entries shared by the controller. As a result, in sRetor, the high flexibility of standard SDN is preserved, allowing the controller to control the switch's behavior when necessary, while offloading some of the forwarding decisions to the switch and reducing the processing pressure on the controller.

The main contributions of this paper are listed as fellow:

- We present the modeling of packet waiting time and controller overhead in an SDN-enabled data center networking.
- We propose a TPDL-based routing scheme for regular SD-DCN on the basis of the modeling and analysis. The proposed method is able to reduce the packet waiting time in switches and controller workload by calculating forwarding paths locally.
- We implement and evaluate sRetor on the Estinet emulation platform and compared it with our previous work and other routing methods. Experiment results show that sRetor reduces the flow start time over 68% and the fail-over time over 84%.

The rest of this paper is organized as follows: Related work section introduces the previous related research work, including data center network routing methods and network overhead reduction in SDN; System model section presents our system modeling on the packet waiting time and controller workload; the system architecture is introduced in sRetor architecture section, followed by the detailed introduction of the proposed forwarding algorithm in Routing algorithms on switches; Numerical results and Evaluation sections present the numerical results and experimental results respectively; Finally, the last section concludes this article.

Related work

Regular data center networking and routing schemes

Many data center network architectures, such as Fat-tree and BCube, have been proposed to improve the performance of data center networks. Most of these new network architectures are built on recursive and iterative approaches. Thus, they tend to have a regular network topology, which means their connecting and addressing are usually in a constant or definite pattern [38]. In addition, for better efficiency and performance, researchers design routing methods corresponding to the structure of these topologies, i.e., topology-aware routing algorithms, achieving more efficient routing leveraging the construction rules of network topologies.

Al-Fares, et al. [9] constructed a large-scale Fat-tree topology for data centers using conventional commercial switches. They also designed a corresponding addressing method by combining the characteristics of the network topology, where the nodes' IP addresses are assigned according to the type, location and other attributes of the nodes. A new two-layer routing method is also proposed, which can directly perform routing based on nodes' IP addresses and connection relationships instead of a complex routing interaction process. The suffix matching method is adopted to forward packets to different up-link interfaces at the edge and aggregation switches based on the host ID of the destination address, making full use of the multi-path feature of the Fat-tree network for load balancing.

Besides, other researchers are still working on improving the routing performance by leveraging the structure of the Fat-tree topology. Liu, et al. [39] proposed a port-based forwarding load-balancing routing method for the Fat-tree topology, which relies on the distinctive addressing scheme of the Fat-tree topology. Edward, et al. [40] proposed the Predictive Equal-Cost Multi-Path protocol in Fat-tree based data center networks, which is inspired by the multi-path diversity of the Fat-tree topology.

In contrast to Fat-tree, BCube [11] is a server-centric data center network architecture, where routing and decisions are made on the server nodes in the network. The topology of BCube could be defined recursively, and numerous network topologies of various sizes can be generated by specifying the number of layers k, which is also a regular network topology. BCube employs the BSR (BCube Source Routing) routing protocol, which utilizes the BCube's topology and multi-path capabilities to accomplish load balancing and fault handling without link-state distribution.

In addition to the classic data center network topologies, such as Fat-tree, BCube and VL2 [41], other regular data center network topologies have been proposed. BCDC [13] is a high-performance server-centric data center network topology based on the crossed cube, a BC network (Bijective Connection network). An n-dimensional BCDC network (B_n) can be defined recursively and is capable of supporting much more network nodes than the Fat-tree topology (with 16-port switches, Fat-tree contains only 1024 servers, while BCDC supports up

to 524,288 servers). The authors also proposed efficient topology-aware routing algorithms for one-to-one, one-to-many, and one-to-all running on BCDC.

LaScaDa [12] uses small port count switches to connect network nodes to clusters with a lower degree, and then connects the clusters to each other following a particular pattern. Therefore, LaScaDa achieves better performance in terms of scalability, average path length, and bisection bandwidth. The authors also propose a new hierarchical row-based routing algorithm to implement packet forwarding in LaScaDa.

Researchers of new architectures mentioned above have designed specific routing techniques for each network topology based on the peculiarities of the connectivity links between nodes. However, these routing methods are not generic and are optimized only for a given topology, which introduces practical deployment challenges. Based on the foregoing observations, we have identified these problems and attempted to resolve them by proposing sRetor. Benefiting from the regular topology description capability of TPDL, sRetor is able to perform routing by leveraging the topological structures of the regular network topology. This routing functionality is generic and works in any data center network topology, addressing the deployment and upgrade difficulties of modern data center networks.

Overhead reduction on software-defined data center networking

The application of SDN in data centers has enabled data center managers to have finer-grained and timely control over data center networks. However, the scalability issue has become a major bottleneck limiting the continued development of software-defined data center networking (SD-DCN). Many overhead reduction methods [42–49] have been developed to improve the efficiency of SD-DCN for overcoming this issue.

In Wang, et al. [42], the authors implemented a dynamic message polling technique on the controller to obtain the state information of the switch. With the dynamic exponential fallback algorithm, the controller can adjust the interval of querying the switch state based on the current state of the switch, therefore reducing the workload and communication overhead of the controller.

Kotani, et al. [44] proposed a method to reduce the CPU load of SDN controllers and control traffic in OpenFlow switches by limiting the number of unimportant Packet-In messages. The authors divided Packet-In messages into three categories: State Change, Flow Setup and Forward, and designed a filter to drop the unimportant Forward messages. Therefore the CPU utilization and bandwidth usage are reduced when heavy flows start, not affecting the expected establishment of other non-heavy flows.

Jia, et al. [45, 46] chose to reduce the runtime overhead of SD-DCN by reducing and balancing the flow table entries, where multi-protocol label switching (MPLS) is adopted for encapsulating routing information. Nodes are selected by their K Similar Greedy Tree algorithm (KSGT) to install flow entries to reduce and balance flow entries among switches. Compared to the schemes that install MPLS flow entries in all nodes, KSGT can reduce about 60% of flow entries.

In Baddeley, et al. [48], the authors proposed μSDN for IoT networks, which applied several approaches to reduce the overhead of SDNs to accommodate lower bandwidth. For example, the μSDN adopts source routing to reduce the overhead at intermediate nodes. Throttle control messages are also adopted to limit duplicate control message requests from consuming extra control bandwidth. Re-using flow table matches/actions reduces flow table entries by merging flow entries with the same destination address.

In Pranata, et al. [49], the authors proposed an overhead reduction framework for SD-DCN, which optimizes SD-DCN at the packet level and flow level to reduce the runtime traffic overhead. At the packet level, the framework ensures that only the first packet of each flow is sent to the controller for reducing redundant Packet-In messages. At the flow level, firstly, the controller mirrors the received flows to the subsequent switches in the forwarding path, to reduce the controller load; secondly, the framework uses MPLS to add forwarding information directly to the data messages to reduce the installation overhead of flow rules. Moreover, to solve the problem of numerous forwarding information entries and data frame length limits, the framework supports splitting the complete MPLS data based on the path length and frame length limits and distributing it to multiple intermediate switches in the forwarding path.

Maliha, et al. [50] focused on the large number of network broadcast packets caused by massive ARP requests in the network. They proposed the ARP-OR framework for efficient APR broadcast reduction and redundancy suppression in SD-DCN. This approach also reduces the bandwidth and computing resource overhead of the control plane.

sRetor addresses the excessive control overhead of SD-DCN from a different perspective. In the conventional SDN networks, the switches need to periodically collect topology information (e.g., by broadcasting LLDP packets to its neighboring nodes), and then report it to the controller. However in sRetor, TPDL is deployed as a priori knowledge to the controllers and switches, allowing the controllers and switches

to obtain a basic consensus of the network topology. Controllers can reserve their limited resources for monitoring topology changes and delivering control messages. Thus controllers are able to support more extensive networks, which makes sRetor more scalable.

System model

A typical architecture of software-defined data center networking is shown in Fig. 1, where the SDN switches are dummy switches and only responsible for executing actions from its flow table. The SDN controller is connected to each switch, either in-band or out-of-band. Here we ignore the details of their secure channel and simplify the communication delay between the controller and switches as constant value t_{RTT}.

In this section, we present the modeling and analysis of both packet delay and controller workload in this SD-DCN architecture.

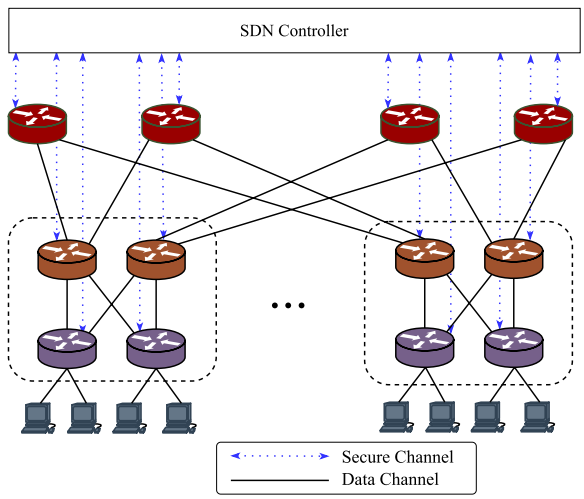


Fig. 1 System Diagram of sRetor scheme

Delay modeling

When a packet n is sent from one switch to another, the point-to-point delay is shown below [51, 52].

$$\tau(n) = t_{queue}(n) + \mathbf{t_{proc}(n)} + t_{trans}(n) + t_{prop}(n) \quad (1)$$

where the $t_{queue}(n)$ is the queuing delay, the $t_{trans}(n)$ is the transmission delay and $t_{prop}(n)$ is the propagation delay. $t_{proc}(n)$ is the processing delay and our focus is to reduce it.

In the traditional SDN solutions [30, 53], the breakdown of processing delay is illustrated in Fig. 2 and its steps are as follows:

- Step 1: Receive a packet from the ingress port.
- Step 2: Look up matched flow entry in the flow table, which results in looking up delay t_{fl}.
- Step 3.1: Execute flow entry if found, which leads to forwarding delay t_{fw}.
- Step 3.2: Send packet to the controller via Packet-In message if no matched entry is found, and it takes $t_{RTT}/2$.
- Step 4 & 5: The controller will make the decision for it and send a Flow-Mod message to the switch. This will produce controller delay t_{ctrl} and another $t_{RTT}/2$.
- Step 6: Execute the newly inserted action to forward this packet, which also needs t_{fw}.

Let $T = t_{RTT} + t_{ctrl}$ be the total delay of communication with the controller, i.e., the total waiting time at the switch. The overall processing delay is defined as follows [54].

$$t_{proc}(n) = t_{fl} + t_{fw} + I_{\alpha}(n) \cdot T$$
$$I_{\alpha}(n) = \begin{cases} 0, & \text{if packet } n \text{ hits the flow table;} \\ 1, & \text{else.} \end{cases} \quad (2)$$

When packet n hits the flow table $I_{\alpha}(n) = 0$, the packet n will be forwarded directly according to the

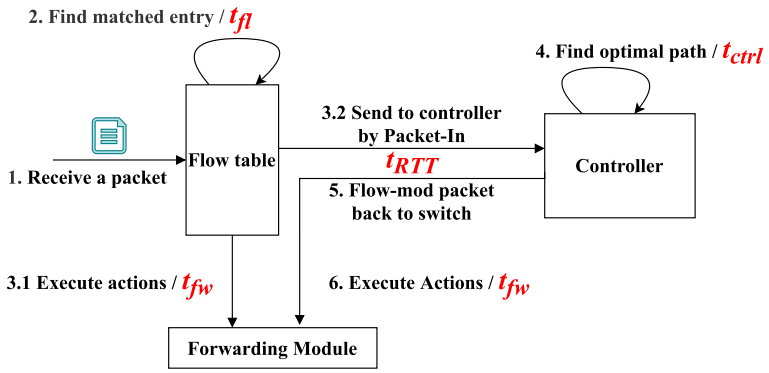


Fig. 2 Processing delay in cRetor(SDN) switches

flow table actions and waiting time T is not needed. While $I_\alpha(n) = 1$, i.e., the packet n did not have any match in the flow table, the packet will be sent to the controller, then the switch will need to wait for T of time. There are several scenarios that will trigger that $I_\alpha(n) = 1$:

- Packet n is the first packet of a flow and there is no entry for this flow in the table.
- The existing next-hop node in the table has failed and the existing related flow entry is invalid.
- Other reasons such as flow entry deletions due to overflow or expiration.

During the waiting duration T, subsequent packets of the same flow may arrive. These packets will be buffered in a pending list and wait until the switch receives the controller's decision as proposed in Pranata, et al. [49].

Let $t = 0$ denote the time when the first packet is sent to the controller. Considering the packets that arrive after the first one and before the switch receives the feedback from the controller, i.e., between $(0, T]$. Their processing time is indicated as follows.

$$\begin{aligned} t_{proc}(n) &= t_{fl} + t_{fw} + I_\alpha(n) \cdot (T - t_n) \\ I_\alpha(n) &= \begin{cases} 0, \text{ if packet } n \text{ hits the flow table;} \\ 1, \text{ else.} \end{cases} \end{aligned} \tag{3}$$

where t_n is the arrival time of packet n between 0 and T, and hence $T - t_n$ denotes the waiting time of the packet. Define the waiting time of packet n between 0 and T as t_{wt}. We assume packets follow a Poisson Point process with a rate λ, the CDF of the arrival time t_n follows [55]:

$$F_{t_n}(t) = 1 - \sum_{i=0}^{N(T)-1} \frac{(\lambda t)^i}{i!} e^{-\lambda t} \tag{4}$$

Where $N(T)$ is the total number of consequent packets that arrives between 0 and T. The CDF of t_{wt} follows:

$$F_{t_{wt}}(t) = 1 - F_{t_n}(T - t) = \sum_{i=0}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)} \tag{5}$$

And the expectation of t_{proc} is shown below,

$$E(t_{proc}) = t_{fl} + t_{fw} + (1 - p_{hit}) \cdot T \cdot \left(1 - \frac{N(T)}{\lambda}\right) \tag{6}$$

Where $p_{hit} = P(I_\alpha(n) = 0)$. As a consequence, to ensure lower processing delay we have to minimise $F_{t_{wt}}(t)$ as below.

$$\begin{aligned} \min \quad & \sum_{i=0}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)} \\ \text{s.t.} \quad & \forall 0 < t \leq T \end{aligned} \tag{7}$$

It is challenging to reduce T in a fixed topology structure. Therefore we propose to reduce the overall processing delay t_{proc}. The forwarding decision (forwarding path for this flow) generated in the controller could be divided into two categories: A) a path that includes current nodes and its subsequent nodes, and B) a new path that does not go via the current node. The probability of the former choice is usually higher than the latter as the controller will only set up subsequent nodes instead of all nodes in the new path. To reduce t_{proc}, we would like to find the path in category A at a local node instead of sending packets remotely and experience controller-switch round-trip time t_{RTT} and t_{ctrl}.

A node should have knowledge of candidate neighbours and destination nodes. However typical SDN switches are dummy switches, which means that they do not collect topology information and therefore they are unable to make forwarding decisions. We propose to adopt TPDL [30] so that the current node can calculate the distance to its neighbours locally, and then make forwarding decisions.

The proposed scheme sRetor is illustrated in Fig. 3. We add a TPDL forwarding step between *Step 3.2* and *Step 5.2*. A packet with $I_\alpha(n) = 1$ will not be forwarded to the controller directly. Instead, it will be sent to the TPDL calculator to look for a local next hop. If this calculation failed either, the controller will get this packet and make a final decision for it.

Let t'_{proc} be the processing time of packet n in sRetor, t'_{proc} and its expectation are shown below,

$$\begin{aligned} t'_{proc}(n) &= t_{fl} + t_{fw} + I_\alpha(n) \cdot (t_{sw} + I_\beta(n) \cdot (T - t_n)) \\ I_\beta &= \begin{cases} 0, \text{ if a next-hop is found locally;} \\ 1, \text{ else.} \end{cases} \end{aligned} \tag{8}$$

$$\begin{aligned} E(t'_{proc}) = t_{fl} + t_{fw} + \\ (1 - p_{hit}) \cdot \left(t_{sw} + (1 - p_{sw}) \cdot T \cdot \left(1 - \frac{N(T)}{\lambda}\right)\right) \end{aligned} \tag{9}$$

Where $p_{sw} = P\big(I_\beta(n) = 0\big)$. The CDF of the wait time in the proposed scheme t'_{tw} will be,

$$F_{t'_{wt}}(t) = p_{sw} + (1 - p_{sw}) \cdot \sum_{i=1}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)} \tag{10}$$

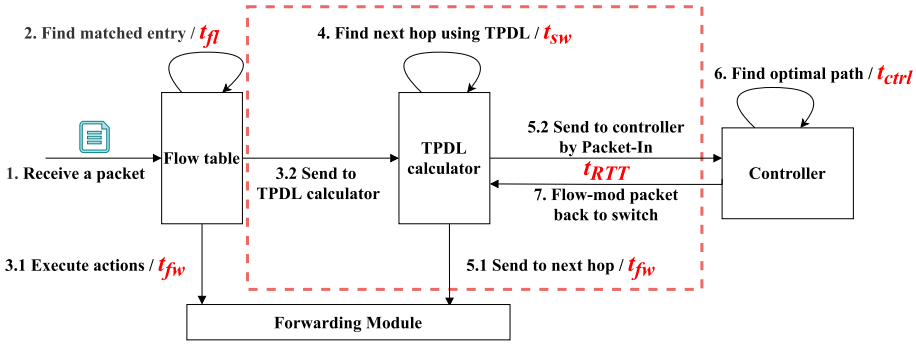


Fig. 3 Processing delay in sRetor switches

To ensure that our scheme achieves lower delay than conventional SDN solutions, we need to fulfill the difference between two schemes $\Delta P(t)$.

$$\Delta P(t) = F_{t'_{wt}}(t) - F_{t_{wt}}(t) > 0 \tag{11}$$

From (5) and (10), we can obtain $\Delta P(t)$,

$$\Delta P(t) = p_{sw} \cdot \left(1 - \sum_{i=0}^{N(T)-1} \frac{(\lambda(T-t))^i}{i!} e^{-\lambda(T-t)}\right) \tag{12}$$

We propose to increase p_{sw}. In the proposed TPDL-based local path-finding algorithm, the p_{sw} is up to 1 without considering the failures, as we could always find the closest next hop in the original topology. However the selected next hop might be unavailable due to the failures. We have to filter out the unavailable neighbors using the dead interval, which is usually ε times of hello interval. The dead interval denotes that a switch will declare a neighbor failed if its hello packet did not arrive within a certain time. Longer dead interval leads to more candidate neighbor nodes and hence higher p_{sw}, while the path success rate could be lower. To trade off between the higher path success rate and higher p_{sw}, the dead interval parameter ε is commonly set to 3 or 4 [56], which ensures a fairly reliable failure detection and higher p_{sw}.

We define the t_{sw} to be the processing time in the TPDL calculator and we aim to reduce t_{sw}. TPDL carries the distance information between any two nodes as described in Jia, et al. [30], so the switches are able to find a neighbour node closest to the destination. The time complexity of TPDL is only related to the number of neighbour nodes, i.e., $O(m)$, where m is the number of neighbouring node.

Controller workload modeling

In the SDN architecture, the centralized controller handles the OpenFlow messages from all switches. *Packet-In* message is one kind of the most common OpenFlow messages generated by the switches when a packet cannot be forwarded locally. Handling Packet-In messages consumes too much computing resources and network bandwidth in the controller [44]. Here we would like to model the controller workload on the basis of the probability of generating Packet-In messages.

As mentioned before, in conventional SDN, the Packet-In message will be generated when $I_\alpha(n) = 1$. Consider these two scenarios: 1) packet n is the first packet of a flow, and 2) link failure(s) occurs in the whole forwarding path. The probability of packet n being sent to the controller via Packet-In message P_{pkt-in} is as follows.

$$p_{pkt\_in}(n) = p_{1st}(n) + (1 - p_{1st}(n)) \cdot \left(1 - (1-q)^m\right) \tag{13}$$

Where $p_{1st}(n)$ is the probability that n is the first packet of a flow, q is the link error rate and m is the forwarding path length. While in the proposed scheme, the Packet-In message is generated when all the available next-hop nodes are failed. Hence P'_{pkt_in} is shown below.

$$p'_{pkt\_in}(n) = p_{1st}(n) + (1 - p_{1st}(n)) \cdot \left(1 - \prod_{i=0}^{m}(1 - q^{c_i})\right) \tag{14}$$

Where c_i is the number of candidate next-hop neighbours, whose distances to the destination are the same and shortest.

$$\begin{aligned} \Delta p_{pkt\_in} &= p_{pkt\_in}(n) - p'_{pkt\_in}(n) \\ &= \prod_{i=0}^{m}(1 - q^{c_i}) - (1-q)^m \geq 0, \\ &\text{given} 0 < q < 1 \text{ and } c_i \geq 1 \end{aligned} \tag{15}$$

Therefore $p'_{pkt\_in}(n) \leq p_{pkt\_in}(n)$, which means that the controllers in sRetor will handle fewer Packet-In messages than in cRetor, and is able to support more extensive SD-DCNs.

sRetor architecture

In this section, we present the overall architecture and components of sRetor. The design goal of sRetor is to reduce the flow establishment time in SD-DCN, while providing dummy switches with basic forwarding capability without support from the controller. Further functions such as load balancing and QoS assurance are left to the controller as it could collect global statistics.

The architecture of sRetor is shown in Fig. 4. This architecture is inherited from the SDN architecture and still consists of the controller and switches, that communicate with each other through the extended OpenFlow protocol. The controller in sRetor is responsible for tracking the real-time status of the entire SDN network and failure information reported by the switches. The controller will find alternative forwarding paths for flows when failures occur. Additionally, the controller also has the ability to distribute TPDL files via the OpenFlow Channel for switch initialization and topology updates.

During the initialization process, the *Topology Manager* in the controller will generate a base network topology with the information from the *TPDL parser*. The switches will report detected failures to the controller in time through the OpenFlow protocol, and the *Topology Manager* will update the connections after receiving these failures information, maintaining the real-time network topology on the controller. The *Routing Calculator* in the controller will recalculate a new feasible path based on the topology information in the *Topology Manager*, and establish a new forwarding path by delivering flow table entries to switches on its way.

After the switch receives the TPDL file delivered by the controller, it also uses the *TPDL Parser* to analyze it for subsequent distance calculation. As shown in Fig. 4, the switch's *forwarding module* gets input from three parts: the *Flow Table*, the *Neighbor Information* and the *Topology Information*. The flow table entries come from the controller and have the highest priority, providing flexible control capabilities equivalent to conventional SDN switches. Neighbor information comes from the static TPDL file and the dynamic *Hello Message Processor*, which monitors the connection status between current and neighboring nodes in real time. Topology Information is extracted from TPDL, providing high-speed distance calculation capability for the forwarding module. The detailed forwarding process is discussed in Routing algorithms on switches section.

With the introduction of TPDL, sRetor empowers the switch with local forwarding decision capabilities, reducing the controller's workload on processing Packet-In messages and topology discovery. This allows a single controller to support more switches in the data center. Furthermore, the retention of SDN components like the flow tables also allows sRetor to have the same centralized control capabilities as SDN and be compatible with the existing SDN ecosystem.

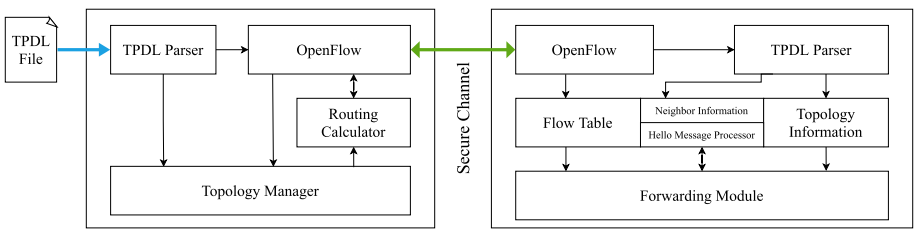
Offloading some of the workloads to the switches could also introduce network security problems to the data planes, such as DDoS attacks. However, many security solutions, such as Mihai-Gabriel, et al. [57] and Varghese, et al. [58] has been proposed for preventing the SDN data plane from being attacked. We believe that most of these solutions will work on sRetor too.

Routing algorithms on switches

In this section, the routing algorithms on sRetor switches are presented and we also give a brief introduction to the switch-level load balancing.

Packet routing process

The switch forwarding process in sRetor has been shown in Fig. 3. This processing flow ensures that the flow table has the highest priority, i.e., the controller still has direct control over the switches, which ensures that the entire network is still under the management of the controller. The TPDL routing calculator can also cache the calculation result by writing its result into the flow table. The flow table is used as a high-speed cache for the calculated



(a) sRetor controller architecture (b) sRetor switch architecure

Fig. 4 System architecture of the sRetor controller and switch

route. The switch will first query whether a cache of the calculation results in the flow table exists; if not, it performs the routing calculation. Thereby we can reduce the number of times of routes calculation and increase the forwarding speed.

Parameter: $rule\_list$: list of distance formulas in a TPDL file.
Require: n_s: source node;
n_d: destination node;
$type(n_s)$: type of source node defined in TPDL;
$type(n_d)$: type of destination node defined in TPDL.
Ensure: distance between node n_s and n_d
1: **for** each $rule \in list$ **do**
2: **if** types of n_s and n_d match $rule$'s requirement **then**
3: $value \leftarrow$ evaluate $rule.condition$ using n_s and n_d
4: **if** $value = true$ **then**
5: **return** $rule.distance$
6: **end if**
7: **end if**
8: **end for**
9: **return** $distance$

Algorithm 1 TPDL distance algorithm (*tpdl_distance*)

Require: n_{cur}: current node;
n_{dst}: destination node;
n_{prev}: previous node of this packet;
$K(Cur)$: node list of current node's neighbours;
$hello\_interval$: interval of hello messages;
t_{now}: current timestamp;
$L(n)$: last hello timestamp of node n.
Ensure: next-hop node of this packet
1: **for** each neighbour node $n_\kappa \in (K(n_{cur}) \setminus n_{prev})$ **do**
2: **if** $t_{now} - L(n_\kappa) < \varepsilon \cdot hello\_interval$ **then**
3: $D_\kappa \leftarrow tpdl\_distance(n_\kappa, n_{dst})$
4: **end if**
5: **end for**
6: find any n^* when $D_{n^*} = min(D_n)$
7: $D_{cur} \leftarrow tpdl\_distance(Cur, Dst)$
8: **if** $D_{n^*} < D_{cur}$ **then**
9: **return** n^*
10: **else**
11: **return** ϕ
12: **end if**

Algorithm 2 Next-hop calculation algorithm on switches

The forwarding path is calculated as shown in Algorithm 2, where the *tpdl_distance*, presented in Algorithm 1, is a function for calculating the distance between nodes leveraging TPDL distance formulas.

Distance in the topology is the main metric for routing calculation in our algorithm. As mentioned in the previous section, we want to place a light workload on the sRetor switches. Collecting network statistics such as available bandwidth and end-to-end delay is costly, thus they are not involved in current routing calculation. However our algorithm can adapt to other metrics with low overhead.

When a packet from the source node n_{src} to the destination node n_{dst} enters the *TPDL Routing Calculator* of the current node n_{cur}, the TPDL Routing Calculator first traverses the set of all available neighbor nodes N that are known via Hello messages. For each available neighbor node $n_\kappa \in \{K(n_{cur}) \setminus n_{prev}\}$, it calculates the distance D_κ from node n_κ to n_{dst} with the help of TPDL's distance formula. Then we find n^* when $D_{n^*} = Min(D_n)$, which means that the node n^* is the closest neighbour to n_{dst}.

This algorithm has the ability to handle direct failures in the network. In the 2nd line of the algorithm, the current time t_{now} is compared to $\varepsilon \cdot hello\_interval$. Neighbour nodes that meet the condition will be the candidate nodes. As a result, the algorithm will only choose the neighbour nodes that were recently reported as the next-hop node.

Load balancing on switches

Due to the regularity and redundancy, data center networks often have many equal-cost paths. Therefore, load-balancing algorithms are essential for data center networks to achieve higher throughput. Two kinds of load balancing in sRetor are expected to be implemented: packet level and flow level load balancing.

A packet-level load balancing mechanism could be implemented as follows: The switches can find all next-hop nodes that are closest to the destination at the same distance. Based on the statistics of the corresponding interface, the candidate next-hop node with the lightest load will be selected. Then the packets will be distributed to different interfaces evenly.

The flow-based load balancing is more sophisticate because the OpenFlow switch is required to *remember* the flows using the flow table. Similar to the packet-based load balancing strategy mentioned above, when the first packet of each flow reaches the switch, the switch will need to find out the next-hop node for this flow. The switch will firstly gather all available shortest paths from the current node to the destination node as candidate paths. Then the switch will select the port that has forwarded the least data packets in the recent time window as the output port of the flow. As shown in the Step 2 of Fig. 5, the switch will then generate a flow entry for this flow, and insert it into the flow table. When the subsequent packets of this flow arrive at the switch, they will be forwarded without further calculation.

In addition, to achieving flow-level load balancing, this method uses the switch's flow table as a cache for routing calculations, reducing the amount of overall calculation, which makes sRetor work efficiently even without specific hardware in switches.

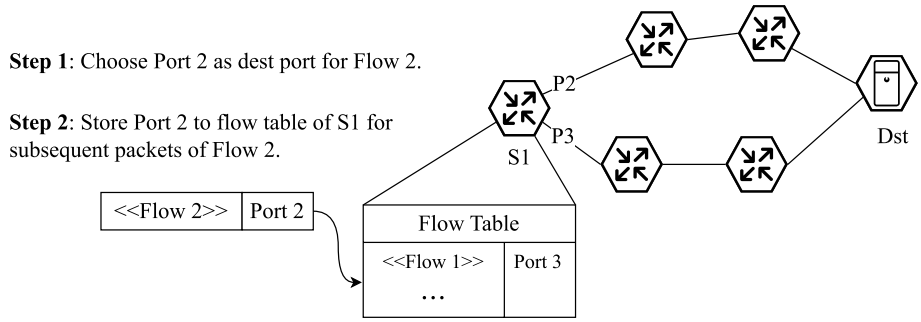


Fig. 5 Flow-level load balancing. The load forwarding result is store into the flow table for subsequent packets in the flow

Fail-over mechanism

In sRetor, a semi-centralized architecture is adopted, so both the switch and the controller have fail-over capabilities. The switches are responsible for handling simple local failures by choosing alternative local next-hop nodes. For more complicated faults, the controller will handle them by distributing flow table entries.

Failures directly associated with the switch itself are mainly handled on the switch, using the TPDL information and the switch's neighbor information for localized fault handling. When a link between a switch and its neighboring nodes in the network fails, the following two types of failures may exist:

- *One of the shortest paths is down, but other ECMP shortest path(s) is/are still up.* This circumstance is common in regular data center networks, e.g., topologies such as Fat-tree often have multiple equivalent paths available. The switch is able to find an alternative shortest neighbor n^* to the destination node satisfying $D_{n^*} < D_{cur}$ using Algorithm 2. Therefore a fast link switchover could be completed on this switch without the need for the controller. Nevertheless, the controller will still learn about this failure through the failure report message from the switch. When the controller regards that this failure has affected the traffic balancing, it can still employ some traffic engineering policies proactively.
- *All of the shortest paths are down.* Thus, the switch will not be able to find a neighbor n^* that is closest to the destination address satisfying $D_{n^*} < D_{cur}$. This situation is usually rare, but it means that this node is not in the global optimal path. Therefore, the switch will stop forwarding locally and send the packet to the controller via a Packet-In message. The controller will determine the best forwarding path using its global topology information.

The improved routing algorithm with the fail-over mechanism is shown in Line 6 to 12 in Algorithm 2. This algorithm also compares D_{n^*} with the D_{cur}, i.e., the theoretical shortest distance from the current node to the destination node. This mechanism is designed to avoid sending packets to detoured paths when failures occur. In addition, this mechanism is effective in preventing forwarding loops as the selected next-hop node is ensured to be no further than the current node.

The sRetor controller is responsible for solving failures that cannot be handled by the switch. Beneficial from the network-wide global view of SDN, the sRetor controller is able to handle concurrent failures and obtain the globally optimal solution. When handling concurrent failures, the fail-over time of sRetor is degenerates into conventional SDN.

Numerical results

In this section, we present our numerical results on the packet waiting time and controller workload mentioned in System model section.

CDF of packet waiting time

We first run simulations on packet waiting time in Eqs. 5 and 10. The simulation parameters are shown in Table 1. This simulation generates flows following the Poisson Point process, and simulates the packet process delay and pending mechanism in switches and the controller.

The CDFs of packet waiting time are illustrated in Fig. 6. We can see that our simulation results shown as histogram align with the analytical models in Eqs. 5 and

Table 1 Simulation parameters on packet waiting time

Parameters	Values
λ	2000
P_{sw}	85%
t_{prop}	300 μs
t_{sw}	100 μs
t_{ctrl}	100 μs
Bandwidth	1Gbps

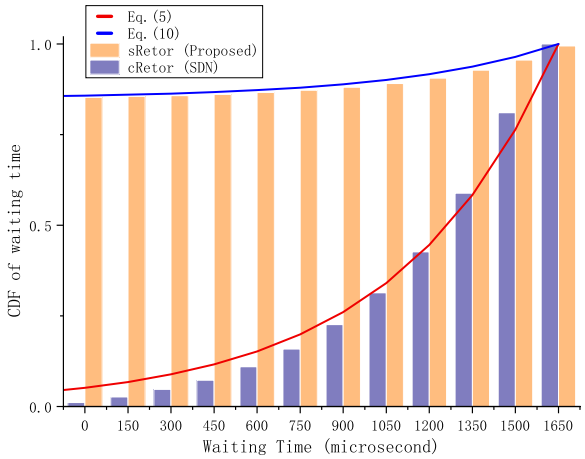


Fig. 6 CDF of packet waiting time

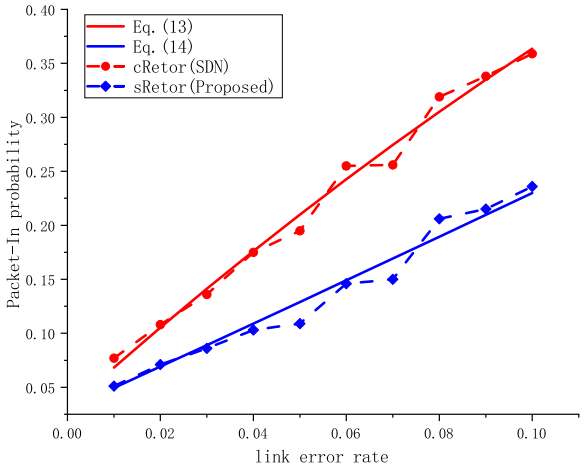


Fig. 7 Packet-In probability with different link error rates

10 that we proposed in Delay modeling section. And the numerical result shows that sRetor has a better performance with less waiting time than cRetor.

Packet-In message probability

We also run simulations on the Packet-In message probability, which shows how many packets will be sent to the controller at various link error rates. The simulation parameters are listed in Table 2.

As illustrated in Fig. 7, there is an obvious difference in Packet-In message probability between sRetor and cRetor, and this aligns with our analysis in Controller workload modeling section. Due to the extra first-packet Packet-In messages and the more alternatives from equal-cost multi paths, sRetor controllers will receive much fewer Packet-In messages from switches. Therefore, the workload of sRetor controllers is lower than controllers in cRetor.

Evaluation

Experimental setup

To evaluate the performance of sRetor, we implemented the sRetor switch on the Estinet network simulation/emulation platform [59] and a sRetor controller on the basis of Ryu [60]. Estinet is a network simulator and emulator that supports both traditional network routing methods (OSPF, BGP, etc.) and OpenFlow SDN, which allows us to compare different routing methods. Ryu is an SDN controller framework written in Python, and lots of previous work has been developed based on it. The controller of sRetor uses the same TPDL parser design, which is developed with the powerful ANTLR language parser generator [61].

Table 2 Simulation parameters on Packet-In message probability

Parameters	Values
P_{1st}	3%
q	1% ∼ 10%
m	6
c_i	[1, 8, 8, 1, 1, 1]
Topology	16-*ary* Fat-tree

We compare sRetor to OSPF, the Fat-tree routing method proposed in Al-Fares, et al. [9] and cRetor in our previous work [30]. The OSPF routing method is powered by the software routing suite Quagga [62], which is a built-in feature of Estinet. The Fat-tree routing method is implemented by ourselves on the Estinet platform according to its proposal. We generate routing tables for each node in the Fat-tree topology following the pattern. The switches load the routing table for prefix/suffix-based forwarding.

We also conducted experiments on another prevalent data center network topology, BCube, to validate the ability of sRetor to work on diverse network topologies. As a server-centric DCN topology, the forwarding decisions in the BCube are made at the servers rather than at the switches, and the switches are low-end commodity switches. Therefore, we have chosen the commonly used 2-tier BCube topology, as the number of forwarding nodes (servers) is close to that of the Fat-tree topology with $k = 4$. This size offers a more comparable evaluation scenario. Other link characteristic parameters remain consistent with the Fat-tree setup. Additionally, we have implemented the BCube Source Routing (BSR) algorithm for comparison. The detailed experimental network parameters are listed in Table 3.

Table 3 Experimental network parameters

Parameters	Values	
Topology	Fat-tree	BCube
Topology Size	$k = 4, k = 8$	2-*level*
Number of switches	20, 80	8
Number of servers	16, 128	16
Link bandwidth	1Gbps	1Gbps
Link Propagation Delay	$1\mu s$	$1\mu s$

Table 4 Flow start time and convergence time on different routing schemes

Topology Size	Schemes	Flow Start Time (ms)	Convergence Time (ms)
4-*ary* Fat-tree	sRetor	1.77	1.79
	cRetor	5.66	1007.08
	OSPF	2.12	50221.33
	Fat-tree	2.24	2.41
8-*ary* Fat-tree	sRetor	1.77	1.78
	cRetor	7.64	1843.48
	OSPF	2.65	52001.71
	Fat-tree	2.26	2.39
2-*level* BCube	sRetor	1.77	1.78
	cRetor	5.70	2001.06
	OSPF	1.23	50213.41
	BSR	2.24	2.24

Flow start time

The flow start time is the end-to-end delay of the first packet being forwarded from the source node to the destination node. Therefore the flow start time t_{flow} is shown as follows.

$$t_{flow} = \sum_{i=1}^{m} \tau_i(n) \tag{16}$$

Where $\tau_i(n)$ is the point-to-point delay of packet n in the ith switch, and m is the number of intermediate switches.

We ran simulations on different routing schemes to evaluate their flow start time. As shown in Table 4, in the Fat-tree topologies, the flow start time of cRetor is substantially higher than that of other routing methods. The communication between the switch and the controller results in a higher flow start time. In contrast, sRetor improves this by making routing decisions locally. Therefore, we achieve a similar short flow start time to other methods such as OSPF and Fat-tree, which both use the lookup table method. The results in the BCube topology also show that sRetor is capable of achieving flow start times comparable to other table-lookup routing algorithms such as OSPF.

Networking convergence time

Another metric related to the packet waiting time is the network convergence time. Due to the separation of the control plane and data plane in the SDN paradigm, the definition of convergence time is also different from that in conventional networks [63]. In this paper, we use the time from the startup of all network devices until all switches are able to communicate with each other as the measure of convergence time.

The simulation results of network convergence time are also shown in Table 4, which illustrates that, compared to traditional link-state routing protocols such as OSPF, the three topology-aware routing methods used in our experiment have substantial advantages in convergence. Both sRetor and Fat-tree/BSR routing methods require almost no additional convergence time. After the switches boot up, they can perform forwarding directly according to the local topology information, which greatly improves convergence speed. Furthermore, it is worth noting that there is no significant difference in the convergence times of these algorithms for networks with different scales. This is because the above-mentioned convergence process is independent of the network scales. This feature makes sRetor more adaptive for large-scale data center networks.

Fail-over time

The fail-over time is also related to the CDF of packet waiting time, due to that failed links lead to table-misses and Packet-In messages in conventional SDN.

We manually create a failure during the simulation. Figure 8 is a snapshot when a failure occurs. We could find that sRetor switches can smoothly be recovered from failures with the capability of local decision-making. The forwarding of packets after the failure has not been affected at all, i.e., the data packets still arrive at the destination node as expected interval, and the delay of the packets keep unchanged. In cRetor, it is obvious that the data packet delay has increased significantly when the failure occurs, from 0.35ms to over 2ms. Another observation is that although there is no packet lost, two data packets arrive at the destination node almost simultaneously due to the extra delay. This observation validates our model that subsequent packets have to wait for the first packet if they arrive between 0 and T. While in OSPF, due to a long time (about 30s) interruption in the network, a large number of data packets are lost.

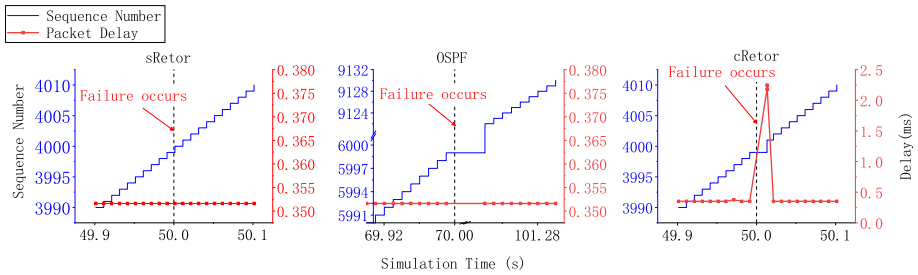


Fig. 8 The Sequence number and delay of packets received in the server side when simple failure occurs

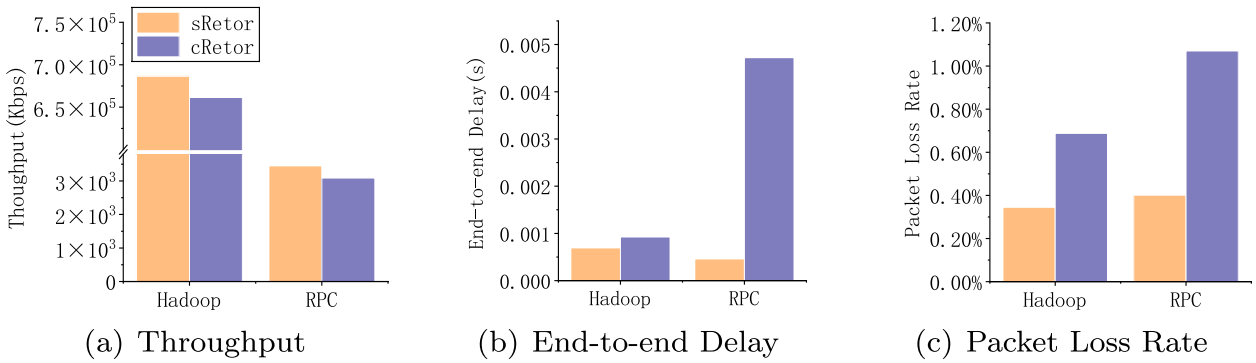


Fig. 9 Throughput, end-to-end delay and packet loss rate of sRetor and cRetor in different real-world traffic patterns

Real-world scenario

We also compare the performance improvement of sRetor in real-world scenarios. The experiments were conducted using the traffic characteristics of the Hadoop cluster from Facebook's data center and the RPC request traffic characteristics from Google's data center provided in Roy, et al. [64]. We implemented a traffic generator for Estinet platform similar to DCTrafficGen [65] by Mellanox and ran experiments in sRetor and cRetor networks. All experiments are conducted in a simulation network with 4-*ary* Fat-tree topology. The experimental results are shown in Fig. 9.

Our experimental results show that sRetor achieves better performance than cRetor in terms of network throughput, end-to-end delay and overall packet loss. Though sRetor is not designed to improve these metrics, the shorter flow establishment time and lower controller workload also contribute to the improvement of the metric. This is because the number of flows in the data center network is enormous, i.e., usually more than 1 million flows arrive at switches per second [66]. The improvement on each flow will finally make a difference to the overall statistics.

Conclusion

In this paper, we modeled the packet waiting time and controller workload and analyzed how to reduce them. Consequently we proposed our topology-aware routing scheme, sRetor, where we applied our previously proposed TPDL to sRetor switches. This enables switches with awareness of the network topology and can work independently when the controller is unavailable.

Numerical and evaluation results show that sRetor has a lower delay in flow start time, network convergence time and fail-over time. Moreover, sRetor decreases the controller workload so that it can support more extensive networks as SDN scales up. Our proposed method provides a reference for future SD-DCN with promising performance to the SD-DCN.

Acknowledgements

The authors would like to express their gratitude to Dr. Ying He for providing help to this paper.

Authors' contributions

Zequn Jia and Yantao Sun gave the main idea of this paper. All authors took part in the discussion and the proposal of the work described in this paper. Zequn Jia wrote sRetor architecture and Routing algorithms on switches sections and conducted the experiments. Yantao Sun wrote the Introduction, Related work, and System model sections and Qiang Liu wrote the rest of sections. All authors reviewed the manuscript.

Funding

No funds have been received from any agency for this research.

Availability of data and materials

Not applicable.

Declarations

Ethical approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 19 February 2022 Accepted: 26 September 2023
Published online: 25 October 2023

References

1. Wang B, Qi Z, Ma R, Guan H, Vasilakos AV (2015) A survey on data center networking for cloud computing. Comput Netw 91:528–547. https://doi.org/10.1016/j.comnet.2015.08.040
2. Ismaeel S, Karim R, Miri A (2018) Proactive dynamic virtual-machine consolidation for energy conservation in cloud data centres. J Cloud Comput 7(1):10. https://doi.org/10.1186/s13677-018-0111-x
3. Amaral M, Polo J, Carrera D, Gonzalez N, Yang CC, Morari A et al (2021) DRMaestro: orchestrating disaggregated resources on virtualized data-centers. J Cloud Comput 10(1):22. https://doi.org/10.1186/s13677-021-00238-6
4. Carrascal D, Rojas E, Alvarez-Horcajo J, Lopez-Pajares D, Martínez-Yelmo I (2020) Analysis of P4 and XDP for IoT programmability in 6G and beyond. IoT 1(2):605–622. https://doi.org/10.3390/iot1020031
5. Suarez Rodriguez AC, Haider N, He Y, Dutkiewicz E (2020) Network optimisation in 5G networks: A radio environment map approach. IEEE Trans Veh Technol 69(10):12043–12057. https://doi.org/10.1109/TVT.2020.3011147
6. He Y, Dutkiewicz E, Fang G, Mueck MD (2015) SNR threshold for distributed antenna systems in cloud radio access networks. In: 2015 IEEE 82nd vehicular technology conference (VTC2015-Fall). pp 1–5. https://doi.org/10.1109/VTCFall.2015.7391145
7. Ferguson AD, Gribble S, Hong CY, Killian C, Mohsin W, Muehe H, Ong J, Poutievski L, Singh A, Vicisano L, Alimi R, Chen SS, Conley M, Mandal M, Nagaraj K, Naidu Bollineni K, Sabaa A, Zhang S, Zhu M, Vahdat A (2021) Orion: Google's Software-Defined Networking Control Plane. USENIX Association, pp. 83–98. https://www.usenix.org/conference/nsdi21/presentation/ferguson. ISBN 978-1-939133-21-2.
8. Xia W, Zhao P, Wen Y, Xie H (2017) A survey on data center networking (DCN): Infrastructure and operations. IEEE Commun Surv Tutorials 19(1):640–656. https://doi.org/10.1109/COMST.2016.2626784
9. Al-Fares M, Loukissas A, Vahdat A (2008) A scalable, commodity data center network architecture. Proceedings of the ACM SIGCOMM 2008 conference on data communication. SIGCOMM '08. Association for Computing Machinery, Seattle, pp 63–74. https://doi.org/10.1145/1402958.1402967
10. Guo C, Wu H, Tan K, Shi L, Zhang Y, Lu S (2008) Dcell: A scalable and fault-tolerant network structure for data centers. Proceedings of the ACM SIGCOMM 2008 conference on data communication. SIGCOMM '08. Association for Computing Machinery, Seattle, pp 75–86. https://doi.org/10.1145/1402958.1402968
11. Guo C, Lu G, Li D, Wu H, Zhang X, Shi Y et al (2009) BCube: a high performance, server-centric network architecture for modular data centers. Proceedings of the ACM SIGCOMM 2009 conference on Data communication - SIGCOMM '09. ACM Press, Barcelona, p 63. https://doi.org/10.1145/1592568.1592577
12. Chkirbene Z, Hadjidj R, Foufou S, Hamila R (2020) LaScaDa: A Novel Scalable Topology for Data Center Network. IEEE/ACM Trans Netw 28(5):2051–2064. https://doi.org/10.1109/TNET.2020.3008512
13. Wang X, Fan JX, Lin CK, Zhou JY, Liu Z (2018) BCDC: A High-Performance, Server-Centric Data Center Network. J Comput Sci Technol 33(2):400–416. https://doi.org/10.1007/s11390-018-1826-3
14. Zhao A, Liu Z, Pan J, Liang M (2019) A Novel Addressing and Routing Architecture for Cloud-Service Datacenter Networks. IEEE Trans Serv Comput 1. https://doi.org/10.1109/TSC.2019.2946164
15. Azizi S, Hashemi N, Khonsari A (2017) A flexible and high-performance data center network topology. J Supercomput 73(4):1484–1503. https://doi.org/10.1007/s11227-016-1836-2
16. Baccour E, Foufou S, Hamila R, Tari Z, Zomaya AY (2017) PTNet: An efficient and green data center network. J Parallel Distrib Comput 107:3–18. https://doi.org/10.1016/j.jpdc.2017.03.007
17. Al-makhlafi M, Gu H, Yu X, Lu Y (2020) P-Cube: A New Two-Layer Topology for Data Center Networks Exploiting Dual-Port Servers. IEICE Trans Commun advpub. https://doi.org/10.1587/transcom.2019EBP3219
18. Feng H, Deng Y, Qin X, Min G (2020) Criso: An Incremental Scalable and Cost-Effective Network Architecture for Data Centers. IEEE Trans Netw Serv Manag 1. https://doi.org/10.1109/TNSM.2020.3036875
19. Habib S, Bokhari FS, Khan SU (2015) Routing techniques in data center networks. In: Khan SU, Zomaya AY (eds) Handbook on data centers. Springer New York, New York, pp 507–532. https://doi.org/10.1007/978-1-4939-2092-1_6
20. Amin R, Rojas E, Aqdus A, Ramzan S, Casillas-Perez D, Arco JM (2021) A survey on machine learning techniques for routing optimization in SDN. IEEE Access Pract Innov Open Solutions 9:104582–104611. https://doi.org/10.1109/ACCESS.2021.3099092
21. Kirkpatrick K (2013) Software-defined networking. Commun ACM 56(9):16–19. https://doi.org/10.1145/2500468.2500473
22. Niranjan Mysore R, Pamboris A, Farrington N, Huang N, Miri P, Radhakrishnan S et al (2009) PortLand: A scalable fault-tolerant layer 2 data center network fabric. Proceedings of the ACM SIGCOMM 2009 conference on data communication. SIGCOMM '09. Association for Computing Machinery, New York, pp 39–50. https://doi.org/10.1145/1592568.1592575
23. Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A (2010) Hedera: Dynamic flow scheduling for data center networks. Proceedings of the 7th USENIX conference on networked systems design and implementation. NSDI'10. USENIX Association, USA, p 19
24. Rojas E, Ibanez G, Gimenez-Guzman JM, Rivera D, Azcorra A (2015) Torii: multipath distributed Ethernet fabric protocol for data centres with zero-loss path repair. Trans Emerg Telecommun Technol 26(2):179–194. https://doi.org/10.1002/ett.2863
25. Wang F, Gao L, Xiaozhe S, Harai H, Fujikawa K (2017) Towards reliable and lightweight source switching for datacenter networks. In: IEEE INFOCOM 2017 - IEEE conference on computer communications. pp 1–9. https://doi.org/10.1109/INFOCOM.2017.8057152
26. Bastam M, Sabaei M, Yousefpour R (2018) A scalable traffic engineering technique in an SDN-based data center network. Trans Emerg Telecommun Technol 29(2):e3268. https://doi.org/10.1002/ett.3268
27. Gonzalez-Diaz S, Marks R, Rojas E, de la Oliva A, Gazda R (2021) Stateless flow-zone switching using software-defined addressing. IEEE Access Pract Innov Open Solutions 9:68343–68365. https://doi.org/10.1109/ACCESS.2021.3077955
28. Abdollahi S, Deldari A, Asadi H, Montazerolghaem A, Mazinani SM (2021) Flow-aware forwarding in SDN datacenters using a knapsack-PSO-based solution. IEEE Trans Netw Serv Manag 18(3):2902–2914. https://doi.org/10.1109/TNSM.2021.3064974
29. Modi TM, Swain P (2022) Intelligent routing using convolutional neural network in software-defined data center network. J Supercomput 78(11):13373–13392. https://doi.org/10.1007/s11227-022-04348-z
30. Jia Z, Sun Y, Liu Q, Dai S, Liu C (2020) cRetor: An SDN-Based routing scheme for data centers with regular topologies. IEEE Access 8:116866–116880. https://doi.org/10.1109/ACCESS.2020.3004609
31. Ghaffari A (2014) An energy efficient routing protocol for wireless sensor networks using a-star algorithm. J Appl Res Technol 12(4):815–822. https://doi.org/10.1016/S1665-6423(14)70097-5
32. SDN/NFV Industry Alliance (2017) Whitepaper on SDN Controller Performance in Data Center Scenario(in Chinese). Technical report, SDN/NFV Industry Alliance
33. Blial O, Ben Mamoun M, Benaini R (2016) An Overview on SDN Architectures with Multiple Controllers. J Comput Netw Commun 2016:1–8. https://doi.org/10.1155/2016/9396525
34. Zhou Y, Wang Y, Yu J, Ba J, Zhang S (2017) Load balancing for multiple controllers in SDN based on switches group. 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, Seoul, pp 277–230. https://doi.org/10.1109/APNOMS.2017.8094139

35. Priyadarsini M, Kumar S, Bera P, Rahman MA (2020) An energy-efficient load distribution framework for SDN controllers. Computing 102(9):2073–2098. https://doi.org/10.1007/s00607-019-00751-2
36. Zhang Y, Cui L, Wang W, Zhang Y (2018) A survey on software defined networking with multiple controllers. J Netw Comput Appl 103:101–118. https://doi.org/10.1016/j.jnca.2017.11.015
37. Isong B, Molose RRS, Abu-Mahfouz AM, Dladlu N (2020) Comprehensive Review of SDN Controller Placement Strategies. IEEE Access 8:170070–170092. https://doi.org/10.1109/ACCESS.2020.3023974
38. Qu G, Chen W Constructing a Large-Scale Data Center Network Structure Using Regular Graphs. In: 2019 IEEE International Conferences on Ubiquitous Computing & Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS). IEEE, Shenyang, pp 809–812. https://doi.org/10.1109/IUCC/DSCI/SmartCNS.2019.00164
39. Liu Z, Zhao A, Liang M (2021) A port-based forwarding load-balancing scheduling approach for cloud datacenter networks. J Cloud Comput 10(1):13. https://doi.org/10.1186/s13677-021-00226-w
40. Nepolo E, Lusilao Zodi GA (2021) A predictive ECMP routing protocol for fat-tree enabled data centre networks. In: 2021 15th international conference on ubiquitous information management and communication (IMCOM). pp 1–8. https://doi.org/10.1109/IMCOM51814.2021.9377396
41. Greenberg AG, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P et al (2009) VL2 - a scalable and flexible data center network. SIGCOMM 39(4):51. https://doi.org/10.1145/1592568.1592576
42. Wang YC (2018) An Efficient Route Management Framework for Load Balance and Overhead Reduction in SDN-Based Data Center Networks. IEEE Trans Netw Serv Manag 15(4):13
43. Iyer AS, Mann V, Samineni NR (2013) SwitchReduce: Reducing switch state and controller involvement in OpenFlow networks. In: 2013 IFIP Networking Conference. IEEE, Brooklyn, pp 1–9
44. Kotani D, Okabe Y (2012) Packet-in Message Control for Reducing CPU Load and Control Traffic in OpenFlow Switches. In: 2012 European Workshop on Software Defined Networking. Darmstadt, pp 42–47. https://doi.org/10.1109/EWSDN.2012.23
45. Jia X, Jiang Y, Guo Z, Wu Z (2016) Reducing and Balancing Flow Table Entries in Software-Defined Networks. 2016 IEEE 41st Conference on Local Computer Networks (LCN). pp 575–578. https://doi.org/10.1109/LCN.2016.96
46. Jia X, Li Q, Jiang Y, Guo Z, Sun J (2017) A low overhead flow-holding algorithm in software-defined networks. Comput Netw. 124:170–180. https://doi.org/10.1016/j.comnet.2017.06.009
47. Obadia M, Bouet M, Rougier JL, Iannone L (2015) A greedy approach for minimizing SDN control overhead. Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft). IEEE, London, pp 1–5. https://doi.org/10.1109/NETSOFT.2015.7116135
48. Baddeley M, Nejabati R, Oikonomou G, Sooriyabandara M, Simeonidou D (2018) Evolving SDN for Low-Power IoT Networks. 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft). IEEE, Montreal, pp 71–79. https://doi.org/10.1109/NETSOFT.2018.8460125
49. Pranata AA, Jun TS, Kim DS (2019) Overhead reduction scheme for SDN-based Data Center Networks. Comput Stand Interfaces 63:1–15. https://doi.org/10.1016/j.csi.2018.11.001
50. Safdar M, Abbas Y, Iqbal W, Umair MY, Wakeel A (2022) ARP Overhead Reduction Framework for Software Defined Data Centers. J Netw Syst Manag 30(3):50. https://doi.org/10.1007/s10922-022-09663-7
51. Ramaswamy R, Weng N, Wolf T (2004) Characterizing network processing delay. IEEE Global Telecommunications Conference, 2004. GLOBECOM '04, vol 3. IEEE, Dallas, pp 1629–1634. https://doi.org/10.1109/GLOCOM.2004.1378257
52. Mathew A, Srinivasan M, Murthy CSR (2019) Packet generation schemes and network latency implications in SDN-enabled 5G C-RANs: queuing model based analysis. 2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC). IEEE, Istanbul, pp 1–7. https://doi.org/10.1109/PIMRC.2019.8904151
53. Darabseh A, Al-Ayyoub M, Jararweh Y, Benkhelifa E, Vouk M, Rindos A (2015) SDDC: A Software Defined Datacenter Experimental Framework. In: 2015 3rd International Conference on Future Internet of Things and Cloud. pp 189–194. https://doi.org/10.1109/FiCloud.2015.127
54. Lin CR, Chen YJ, Wang LC (2017) Handoff Delay Analysis in SDN-Enabled Mobile Networks: A Network Calculus Approach. 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall). IEEE, Toronto, pp 1–5. https://doi.org/10.1109/VTCFall.2017.8288202
55. Muhizi S, Shamshin G, Muthanna A, Kirichek R, Vladyko A, Koucheryavy A (2017) Analysis and performance evaluation of SDN queue model. In: Koucheryavy Y, Mamatas L, Matta I, Ometov A, Papadimitriou P (eds) Wired/Wireless internet communications. Springer International Publishing, Cham, pp 26–37
56. Vidalenc B, Noirie L, Ghamri-Doudane S, Renault E (2013) Adaptive failure detection timers for IGP networks. In: 2013 IFIP networking conference. IEEE, Brooklyn, pp 1–9
57. Mihai-Gabriel I, Victor-Valeriu P (2014) Achieving DDoS Resiliency in a Software Defined Network by Intelligent Risk Assessment Based on Neural Networks and Danger Theory. In 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI). IEEE, Budapest, pp 319–324
58. Varghese JE, Muniyal B (2021) An Efficient IDS Framework for DDoS Attacks in SDN Environment. IEEE Access Pract Innov Open Solutions 9:69680–69699. https://doi.org/10.1109/ACCESS.2021.3078065
59. Wang S-Y, Chou C-L, Yang C-M (2013) EstiNet openflow network simulator and emulator. IEEE Commun Mag 51(9):110–117. https://doi.org/10.1109/MCOM.2013.6588659
60. Ryu SDN Framework. https://ryu-sdn.org/. Accessed 30 Apr 2021
61. ANTLR (ANother Tool for Language Recognition). https://www.antlr.org/index.html. Accessed 30 Apr 2021
62. Quagga Routing Suite. https://www.nongnu.org/quagga/index.html. Accessed 30 Apr 2021
63. Abdallah S, Kayssi A, Elhajj IH, Chehab A (2018) Network convergence in SDN versus OSPF networks. In: 2018 fifth international conference on software defined systems (SDS). pp 130–137. https://doi.org/10.1109/SDS.2018.8370434
64. Roy A, Zeng H, Bagga J, Porter G, Snoeren AC (2015) Inside the Social Network's (Datacenter) Network. Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. SIGCOMM '15. Association for Computing Machinery, New York, pp 123–137. https://doi.org/10.1145/2785956.2787472
65. DCTG Data Center Traffic Generator Library (2018) https://github.com/Mellanox/DCTrafficGen. Accessed 30 Apr 2021.
66. Benson T, Akella A, Maltz DA (2010) Network Traffic Characteristics of Data Centers in the Wild. Proceedings of the 10th Annual Conference on Internet Measurement - IMC '10. ACM Press, Melbourne, p 267. https://doi.org/10.1145/1879141.1879175

Publisher's Note