oFIQUIC: Leveraging QUIC in OSPF for seamless network topology changes

Nicolas Rybowski*†§, Cristel Pelsser†, Olivier Bonaventure*†
*WEL Research Institute, Wavre, Belgium
†UCLouvain (*ICTEAM*), Louvain-la-Neuve, Belgium
firstname.lastname@uclouvain.be

***Abstract*—Link state-routing protocols such as OSPF and IS-IS are used in most if not all Internet Service Provider and enterprise networks. They both rely on flooding to distribute the network topology to all routers. Upon topology changes, all routers update their forwarding tables asynchronously which leads to transient events such as micro-loops and packet losses.**

We propose two improvements to OSPF in an extension called oFIQUIC. First, we use QUIC to exchange routing information between neighboring routers. Second, we revisit the OSPF flooding process. Instead of relying entirely on flooding to distribute topology changes, we establish secure remote QUIC sessions with distant OSPF routers to inform them of topology changes. This enables oFIQUIC to prevent transient loops by ordering the updates of the forwarding tables of all routers after a topology change. We add oFIQUIC to the BIRD implementation of OSPF. Our evaluation demonstrates that oFIQUIC prevents loops and converges quickly in different topologies.

***Index Terms*—OSPF, IS-IS, routing protocols**

I. Introduction

Modern networks are largely over-provisioned [1], [2]. They are designed to absorb peak hours traffic and large instantaneous spikes. During the COVID19 pandemic, most networks supported the significant changes in traffic patterns caused by massive remote work [3]. The extra bandwidth present in today's networks enables to cope with equipment failures by rerouting traffic on alternate paths. It also permits network maintenances in a make-before-break manner where load is directed away from the device to be maintained and then returned to the original path after the operation. Further, to reduce the energy consumption of networks one can envisage to temporarily turn some links down and re-enable them when needed [2].

OSPF [4] and ISIS [5] are routing protocols commonly used by ISP and enterprise networks to determine the forwarding paths within their network. These protocols are link-state routing protocols. With these two protocols, all routers build a directed graph that represents the entire network topology. Using Dijkstra's shortest path algorithm, each router can locally compute its forwarding table. However, after a topology change, micro-loops can occur while the protocol is re-converging. Such micro-loops frequently appear in operational networks [6]. To mitigate these micro-loops, router vendors tune their implementations of OSPF and IS-IS to reduce their convergence time [7] or rely on fast-reroute techniques [8], [9] such as Loop-Free Alternates [10], [11], Remote LFA [12] and more recently the TI-LFA [13]. The fast reroute techniques allow to quickly reroute packets around the failed link, while the underlying routing protocol converges to an updated network state. These solutions mitigate micro-loops or their effect but do not remove their occurrences. Researchers have proposed several techniques to prevent those micro-loops [14], [15] but they have not been adopted by the IETF. For pure IP networks, the IETF adopted a framework for ordering forwarding table updates [16] and then a timer-based solution [17]. The latter solution increases the convergence time of the protocol. Here we adopt an approach inspired from the ordering of forwarding table updates [15].

In parallel, most Internet applications have evolved from using TCP to TLS over TCP [18]. Some applications have started to complement TCP by QUIC. QUIC [19] was initially designed with HTTP in mind. It runs above UDP and includes the same security features as TLS 1.3 [18]. From a reliability viewpoint, QUIC is on par with modern TCP implementations [20]. In addition to HTTP/3, QUIC is used to support DNS [21], iCloud private relay [22], [23] and the IETF explores its utilization for live-streaming, gaming and media conferencing [24]. OSPF and IS-IS still use their own transport layer despite the security benefits that secure transports such as TLS over TCP or QUIC could bring. Integrating routing protocols with QUIC is a work still in early development stages [25].

In this paper, we propose a new approach to prevent micro-loops upon topology changes in pure IP networks. Instead of relying on vanilla flooding to distribute link state packets and letting routers update their forwarding tables asynchronously, each router informs routers of a topology in a specific order. This approach requires setting up temporary OSPF sessions with distant routers. To ensure that routers exchange this information securely, we use QUIC. In a nutshell, when activating or disabling a link, the head-end router derives an ordering for distant routers to update their forwarding tables that guarantees the absence of micro-loops. It then follows this ordering to contact each router and send the new link-state packet. Although our prototype works with OSPF, a similar approach could be used with IS-IS.

This paper is organized as follows. We first explain in Section II why re-configuring topologies without introducing

§This work has been partially supported by the Walloon Region as part of the funding of the FRFS-WEL-T strategic axis.

ISBN 978-3-903176-63-8 © 2024 IFIP

transient events such as micro-loops is important. Section III provides insights on the oFIQUIC design. We explain how we integrated QUIC in an OSPFv3 implementation, how updates ordering is derived and how the ordered FIB update process is implemented. We present in Section IV the DUNE framework we developed for precise emulation measurements. Its ability to perform this task is showcased by measuring micro-loops triggered by OSPF in a simple topology. We then evaluate in Section V the behavior of our oFIQUIC prototype on different topologies by leveraging the DUNE framework. Finally, we conclude in Section VI.

II. Motivation

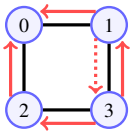
The topology of an enterprise or ISP network is not static. There are several reasons why network operators change the topology of their network. First, network operators may need to add or remove physical links [1], [26], [27]. This is a relatively rare event in network backbones [1] but a more frequent one at the edge to add or remove customer or peering links. The former events are handled by the link-state routing protocols while the latter mainly affect BGP. In the coming years, it is possible that backbone networks will become more dynamic. If the IP network runs above a managed optical network, the network operator can easily add a new link between two routers by reconfiguring the optical network. Furthermore, backbone networks are heavily over provisioned [1]. Researchers have proposed to disable lightly used links during non-peak periods to minimize the energy consumption of backbone networks [28].

A second reason to modify a network topology is when the bandwidth of some links is increased or decreased. Most network operators configure their OSPF metric in function of the link bandwidth. Any change in link bandwidth will result in a topology change. These bandwidth changes are often performed by reconfiguring the underlying optical network. For network operators that manage both the IP and the optical network, those changes can be frequent.

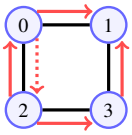
A third reason to modify the topology of a backbone network is for traffic engineering purposes. Researchers have proposed various techniques to optimize the OSPF metrics [29], [30] to redirect flows away from congested links. These techniques usually start from an estimation of the traffic matrix and produce a set of OSPF metrics which can be changed to reduce congestion. Some techniques also minimize the number of metrics that need to be changed [31].

The last cause for topology changes are the link failures. These events are frequent in large networks [26], [32].

Upon a topology change, OSPF routers react by flooding updated Link State Advertisements (LSAs) reflecting the new network state. This flooding process is, by design, unordered. After a topology change, all routers update their FIB in a random order which mainly depends on the arrival time of the new LSA on each router and the time required to process it. This can lead to transient states during which the global data-plane is unstable due to transient loops. Such situation is illustrated in Figure 1a, where all links have a unit metric.



(a) Metric increment of $1 \rightarrow 0$



(b) Metric increment of $0 \rightarrow 1$

Fig. 1: Transient forwarding loops on a simple topology. Each link is configured with a metric of 1. Pre-convergence ECMP paths are show in red. Dotted lines show updated post-convergence paths.

Red arrows show the paths used by Node 3 to reach the other nodes. For example, Node 3 reaches Node 0 through Nodes 1 and 2. Now, consider that we reconfigure the link between Nodes 0 and 1 to use a metric of 10. Node 1 updates its FIB and redirects the packets destined to Node 0 via Node 3, as shown by the dotted line in Figure 1a. However, Node 3 did not update its FIB yet, hence bouncing the packets back to Node 1. This transient loop, also called micro-loop, will last until Node 3 receives the new router-LSA of Node 1 and updates its FIB accordingly. In the best case, packets loop for a very short duration and this only creates some reordering. However, in the worst case, the loop can last long enough so that the TTL of the forwarded packets reaches 0. In that case, the packets are dropped, and the loop becomes a transient blackhole. In the above example, the loop can be prevented provided that Node 3 updates its FIB **before** Node 1. A similar micro-loop happens in Figure 1b between Nodes 0 and 2. It is avoided if Node 2 updates its FIB **before** Node 0.

Researchers have shown that it is possible to prevent all transient loops after a topology change by ordering the FIB updates on all the routers that are affected by the change [14], [15], [33]–[35]. The ordering depends on the type of topology change (metric increase or decrease, addition or removal of a node, ...). The IETF and router vendors discussed these ordering techniques [36] and came up with two possible approaches. A first approach is to use a dedicated protocol to compute the ordering of the FIB updates [16]. However, this solution has never been specified for OSPF or IS-IS. Later, the IETF adopted a simpler solution [17] that delays the FIB updates on some nodes to prevent micro-loops. Unfortunately, this approach only works for a subset of the changes where a link metric increases.

III. The design of oFIQUIC

A key design choice of link-state routing protocols such as OSPF and IS-IS is that routing sessions are only established between direct neighbors. An OSPF router never interacts directly with a router that is several hops away. It only exchanges its link state information with its direct neighbors. oFIQUIC changes this assumption. As explained in the previous section, it is possible to prevent micro-loops by ordering the updates of the FIB of the routers. To realize this ordering, an oFIQUIC router can create a temporary routing session with a distant router to request it to update its FIB for a specific change.

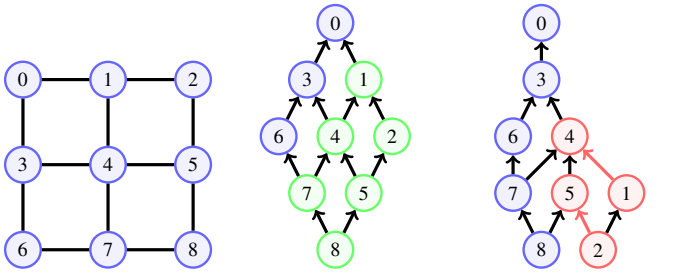
oFIQUIC uses QUIC to create this temporary routing session for two reasons. First, QUIC is implemented entirely in user space. This implies that a router implementation can easily add the required libraries to support QUIC without requiring any kernel change. Second, QUIC fully supports TLS 1.3. In particular, and even if this is not important for HTTP/3, QUIC supports client certificates. In a network, routers should only accept routing information from trusted routers. oFIQUIC requires the network operator to assign certificates to each OSPF router and validates the client certificates when a routing session starts. This enables oFIQUIC to securely establish routing sessions between direct and remote neighbors. This provides as good security as state-of-the-art techniques used by OSPF implementations (keyed MD5 [4], keyed SHA [37] or IPSec tunnels [38]). In addition, QUIC provides faster recovery from packet losses than vanilla OSPF loss recovery.

The OSPF specification defines a neighbor discovery method, called *HELLO protocol*[1]. The first part of this protocol ensures two-way connectivity by exchanging *Hello* messages. OSPF nodes embed their *router ID* in their *Hello* messages. They also reflect *router IDs* present in *Hello* messages they receive. When a node sees its own *router ID* in the *Hello* message originated by its neighbor, it transitions to the `2-WAY` state of the OSPF neighbor Finite State Machine (FSM). This verification ensures that both nodes are able to exchange messages with their neighbor.

To initiate the QUIC session between two nodes, we slightly modify the behavior of the OSPF neighbor FSM. We keep the classical neighbor discovery mechanism based on *Hello* messages exchanged over IP. When the state machine reaches the `2-WAY` state, the node with the smallest *router ID* asynchronously initiates a QUIC session with its neighbor. Transition to `EXSTART` state is blocked until a QUIC stream is correctly established between both nodes. Once the QUIC session is connected and the stream established, we change the communication socket from the IP one to the QUIC stream one, and transition to the `EXSTART` state. This method redirects all subsequent unmodified OSPF messages over the QUIC routing session. The IP socket is kept open in case of neighbor failure. Upon transition to the `DOWN` state, we fall back on the IP socket so that the neighbor discovery can restart afterwards. Our use-case is limited to point-to-point OSPF links. We leave for further work the support for other types of OSPF links.

A. Ordering FIB updates

Previous works explored multiples methods to order IGP updates. François et al. [15], [17] proposed a solution where each IGP node estimates the instant at which it has to update its FIB after receiving an IGP update. This method is solely based on timers and temporal estimations of neighbors behavior without explicit feedback. Clad et al. [33], [34] and François et al. [14] explored another direction to suppress transient events in case of topology reconfiguration. Their method relies on ordered sequences of metric updates ensuring loop-free reconfiguration.



(a) 3×3 grid. (b) Pre-convergence $rspDAG_{1\to0}(0)$. (c) Post-convergence $rspDAG_{1\to0}(0)$.

Fig. 2: Simple 3×3 grid topology. Each link metric is unitary. Figure 2b shows the rspDAG rooted at Node 0 before the metric increase of link (0,1) while Figure 2c represents the rspDAG after the change. Green nodes in Figure 2b represent the sub-$rspDAG_{1\to0}(0)$ rooted at Node 1. Red arrows in Figure 2c indicate inverted edges w.r.t. the pre-convergence rspDAG.

Our FIB updates ordering is based on the method defined by François et al. [15]. It is defined as follows when the metric of link $X \to Y$ needs to be incremented. Node *X* recomputes its reverse shortest path Directed Acyclic Graph (rspDAG) rooted at *Y*, denoted $rspDAG_{X\to Y}(Y)$. Figure 2b illustrates $rspDAG_{1\to0}(0)$ computed by Node 1 for the network shown in Figure 2a. The set of nodes belonging to the sub-$rspDAG_{X\to Y}(Y)$ rooted at node *X* might be affected by the link $X \to Y$ reconfiguration. Green nodes in Figure 2b represent the sub-$rspDAG_{1\to0}(0)$ rooted at Node 1. Router *X* computes the depth of each node in sub-$rspDAG_{X\to Y}(Y)$ and orders them from the deepest node to the nearest one. If multiples nodes are found at a same depth, they are ordered by increasing OSPF *router ID*. For example, the ordered list computed on $rspDAG_{1\to0}(0)$ for the topology of Figure 2a is $[8, 5, 7, 2, 4]$. As updates of every node in the rspDAG are explicitly ordered, we call this method a ***total*** ordering. Upon bidirectional re-configuration (or failure) of link (X, Y), both Node *X* and Node *Y* compute their rspDAG, respectively $rspDAG_{X\to Y}(Y)$ and $rspDAG_{Y\to X}(X)$. We define a *scheduling node* as a node performing ordered convergence. In the situation depicted above, Node *X* and Node *Y* are scheduling nodes.

Not every node belonging to the sub-$rspDAG_{X\to Y}(Y)$ will trigger a micro-loop. Solely couples of consecutive nodes present on post-convergence and pre-convergence path(s) in the reverse direction trigger loops of size two. Predicting such inversion requires also computing the post-convergence rspDAG. Figure 2c shows the post-convergence $rspDAG_{1\to0}(0)$ computed by Node 1. The depth change of Node 1 from one to three in the rspDAGs indicates a path modification. The same behavior is observed for Node 2 whose depth changes from two to four. Comparing their successors in the post-convergence $rspDAG_{1\to0}(0)$ with their predeces-

[1]RFC2328, Section 10.3. [4]

sors in the pre-convergence $rspDAG_{1\to0}(0)$ reveals candidate loops between Nodes 5 and 2 and Nodes 4 and 1. The final ordering is computed by comparing pre-convergence and post-convergence ECMP paths. We iterate on each candidate node in the pre-convergence sub-$rspDAG_{X\to Y}(Y)$, starting from the deepest node with the lowest ID. When multiple nodes are found at the same depth, we order them by increasing router ID. For a given candidate, we iterate on its pre-convergence nexthops. We then iterate on the nexthops of each node of this list in the post-convergence rspDAG. When such nexthop is our current candidate, we found a loop. If the candidate is not already present in the ordered list, it is appended.

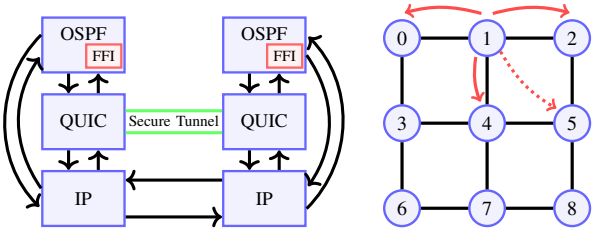
For the topology defined in Figure 2, this generates the following exploration. The candidate list, defined by the sub-$rspDAG_{1\to0}(0)$, is $[8, 5, 7, 2, 4]$. Node 8 is considered first. In the pre-convergence $rspDAG_{1\to0}(0)$, its first nexthop towards Node 0 is Node 5. Nexthop of Node 5 in the post-convergence rspDAG is Node 4, Node 5 is thus ignored. The next nexthop of Node 8 is Node 7. Nexthops of Node 7 towards Node 0 are unchanged before and after the metric update, hence Node 8 is ignored. We now consider Node 5 whose first pre-convergence nexthop is Node 2. In the post-convergence rspDAG, one of the nexthops of Node 2 is Node 5, we just found a loop. Node 5 and 2 are appended to the ordered list. The next nexthop of Node 5 in the pre-convergence rspDAG is Node 4, which does not trigger micro-loops. Nodes 7 and 2 are then explored but none of them trigger micro-loops. The last explored node is Node 4. Its first pre-convergence nexthop is Node 1, whose post-convergence nexthop is Node 4. We found the second loop, Node 4 is appended to the ordered list but not Node 1 as it is the scheduling node. It is implicitly the last node of the ordered list, as it triggers its own convergence once each node of the ordered list has converged. The final ordering collected is $[5, 2, 4]$. This method, that we call ***partial*** ordering, reduces the number of nodes to explicitly contact after a change at the cost of additional computations. Due to space limitation, the ordering in case of metric decrement (or link recovery) is not discussed in this paper.

B. oFIQUIC prototype

First, we extend the BIRD implementation of OSPFv3 [39] with QUIC [40]. This extension allows instantiating secure routing sessions between OSPF nodes. Second, we leverage QUIC to contact remote OSPF nodes in an ordered manner upon topology change. We define a *distant node* as a node being explicitly contacted via a QUIC tunnel by a scheduling node during ordered convergence. Nodes 5, 2 and 4 are distant nodes in the situation described above.

Both ordering methods described in Section III-A are implemented in a Rust crate. A small Foreign Function Interface (FFI) is built on top of the crate and the whole library is statically compiled, so that it is directly embedded in BIRD as depicted in Figure 3a.

Upon link failure requiring the removal of an adjacency or a metric increment pushing the traffic away from a link, ordered FIB updating is triggered on both nodes adjacent



(a) High level design. (b) Scheduling Node 1 establishes secure tunnels with distant nodes.

Fig. 3: OSPF over QUIC design. Figure 3b depicts in red oFIQUIC routing tunnels from Node 1 toward its neighbors. The dotted tunnel is a remote routing tunnel established by the scheduling Node 1 towards its distant Node 5.

to the link. The first step is feeding the entire OSPF Link State DataBase (LSDB) to the ordering module. The module returns an ordered list of nodes to contact, as described in Section III-A. The scheduling nodes open asynchronously a QUIC connection toward each node in the ordered list as depicted by the red dotted line in Figure 3b. A callback system allows maintaining the ordering while sending the updated LSA as soon as QUIC sessions are connected. When the QUIC stream is established with the distant node, the scheduling node verifies whether the predecessor of the distant node in the ordered list already has converged. If it is not the case, a callback is registered with the predecessor data. If the predecessor already has converged, or if the distant node is the first one of the ordered list, the scheduling node immediately sends its LSA though the QUIC session. Upon reception, the distant node directly injects the LSA in its LSDB, triggers the shortest path computation, and updates its RIB and then its FIB. The distant node replies to the scheduling node with a convergence notification through the QUIC tunnel. Unlike vanilla OSPF, the distant nodes do not flood this special LSA. Upon reception of this notification, the scheduling node closes the QUIC connection and triggers the callback of the next distant node in the ordered list. Once the scheduling node has contacted all the nodes of the list, it can finally safely update its own FIB. It then increments the sequence number of the new LSA and floods it through the complete topology. This last step ensures that every node has the correct LSA in its LSDB. Receiving this LSA will not trigger any FIB update resulting in micro-loops thanks to our explicit ordering method. The topology update is finished after this step. This is, to our knowledge, the first real implementation of oFIB [16]. The ordering module is written in 1328 lines of Rust code, tests included.

We currently only consider single events at a time, which has two main implications. First, we have at most two scheduling nodes, one at each end of the updated link. Second, distant nodes are ensured to be reachable during ordered convergence. Indeed, if a node is unreachable, this would imply another concurrent event; which is not possible due to our working hypthesis of single events.

IV. Measuring micro-loops during OSPF convergence

Evaluating routing protocols implementations requires either a physical infrastructure, i.e., a bunch of interconnected routers, or a virtualized one. Using a physical infrastructure with a fixed number of routers limits the size of the topologies on which the prototype may run. Another limitation is that software runs in real-time with little control on the operating system, making the timing measurement less reproducible from one run to another. Virtualized infrastructures are classified in three main categories, (*i*) simulators, (*ii*) emulators and (*iii*) hybrid approaches combining the best of both worlds. Discrete-time simulators such as ns-3 [41] or OMNeT++ [42] provide, by design, the best reproducibility and timing measurement capabilities. Their main drawback is that they require models of protocols and do not execute real protocol implementations. Emulators like mininet [43] execute unmodified implementations in real-time but do not allow reproducible timing measurements. Such tools rely on the underlying kernel for resources allocation. For example, different emulated routers can run on the same CPU core or an emulated router may be moved by the Linux scheduler from one core to another while running time-critical code. Such undesired events may introduce meaningless delays in timing measurements of the prototype under test. Hybrid approaches such as Shadow [44], ns-3-DCE [45] or SimBricks [46] allow running unmodified protocol implementation on discrete-time simulators but still present major limitations. The main ones are (*i*) requiring network protocol models, hence limiting the flexibility of the tool when the tested implementation leverages non-modelled protocols, e.g., Shadow [44] leverages models of TCP and UDP, (*ii*) using sometimes old network stacks, which limits the technologies usable in tested prototypes [47], (*iii*) no or little control on resource allocation while emulating unmodified real-world implementations and (*iv*) scalability issues while simulating medium to large scale topologies.

At the time of writing this document, none of the tools discussed above is mature enough to fit our use-case, i.e., measuring precise timing of events triggered by low level distributed routing protocol implementations in a reproducible manner. To that end, we developed the lightweight Distributed Micro Network Emulation (`DUNE`) framework, by leveraging Linux kernel resource isolation and allocation capabilities.

The remaining of this section is structured as follows. First, we describe in Section IV-A the architecture of `DUNE`. Then, we showcase in Section IV-B the capabilities of `DUNE` by measuring micro-loops during the convergence of vanilla OSPF after a modification on a simple topology. We confirm that `DUNE` is appropriate to detect micro-loops and measure their duration.

A. Distributed Micro Network Emulation (`DUNE`) framework

`DUNE` aims at providing a lightweight emulation framework allowing relatively precise timing measurements with unmodified network protocols implementations. It leverages native abilities of the Linux kernel to provide resource allocation and isolated network stacks. Thanks to this design, tested prototypes are able to use network protocol implementations of the underlying kernel. This removes kernel version or protocol model restrictions that are typical of hybrid simulators.

Figure 4 showcases how a simple triangle topology is defined within the `DUNE` framework. Network nodes are represented by isolated processes and a dedicated network stack, provided by a Linux network namespace. In order to isolate such processes, hyper-threading is physically disabled and the Linux kernel is configured using the `isolcpus` parameter. Every CPU core, except one on which the kernel runs, is unusable by the Linux scheduler. Network node processes are then explicitly pinned to cores belonging to a same NUMA node. Such configuration removes unwanted delays introduced by the Linux scheduler and allows improved temporal proximity for the processes of a given network node. Links are emulated using virtual Ethernet (`veth`) pairs whose properties, i.e., latency and bandwidth, are configured with `tc`.

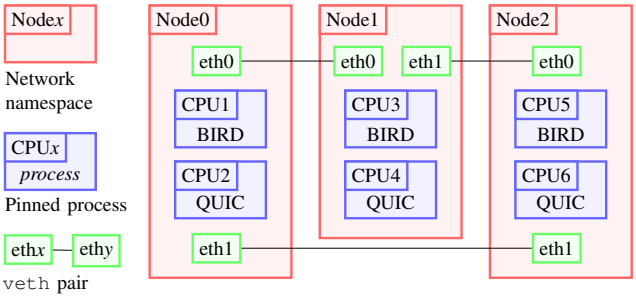


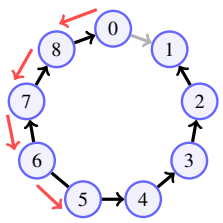
Fig. 4: `DUNE`-based oFIB prototype deployment on a simple triangle topology.

`DUNE` provides orchestration and experiment automation primitives to the users. The framework takes as input a topology description and outputs the commands required to configure the provided topology on off-the-shelf servers. It also launches the pinned network node processes. The framework renders user-defined templates based on the provided network topology. For example, users may provide a single configuration template of a routing daemon and `DUNE` generates the configuration of each router. Finally, the framework produces binaries of the implementation under test based on a build environment defined by the user. All in all, `DUNE` allocates hardware and software resources so that unrelated delays are suppressed. It greatly decouples a protocol implementation to test from its underlying testbed.

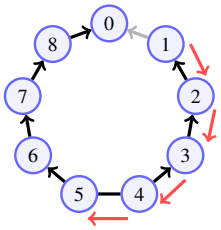
B. Experimental framework and evaluation methodology

In a given topology, each link is configured as a point-to-point link whose link-LSAs are disabled. We configure a unitary metric on each link and set the *HELLO* timer to 5 s. BIRD's internal timer is configured to 10 ms. We use unnumbered interfaces and assign an IPv6 address to the loopback of each router, which is the sole prefix they advertise. Links latency is configured to 5 ms. After starting OSPF on each node, the experiment is left untouched during 30 s to let

OSPF converge. Then, the metric of the target link is raised from 1 to $max\_metric - 1$. We let OSPF re-converge for 30 s then stop the experiment. We define the *convergence duration* as the duration between the instant at which the reconfiguration is effective on the first node and the instant of the last FIB update in the network. During each experiment, we listen to the netlink socket of the FIB on each node. Every received message is associated with an epoch. During post-processing, the collected FIB updates are ordered by epoch, providing a global timeline of updates for the whole topology. At each step of the timeline, we incrementally build the FIB of the node that received the update. Hence, we explore the global forwarding plane state, update by update. After the initial convergence, we rebuild all ECMP routes from each node to each node. This method allows the detection of micro-loops. We test each set of parameters defining an experiment ten times. The experiments described in this section are performed on a single server embedding dual `Intel Xeon E5-2687W v3@3.10GHz`, for a total of 20 physical cores. The server is running Debian 12.



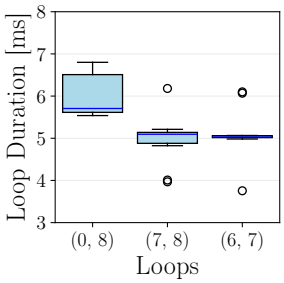
(a) Link $0 \rightarrow 1$ configuration.



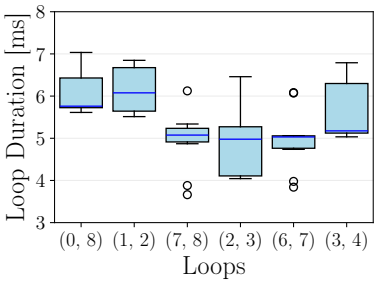
(b) Link $1 \rightarrow 0$ configuration.

Fig. 5: Simple ring of 9 routers. Each link metric is unitary. Paths from each node toward Node 1 are shown before (black) and after (red) the metric reconfiguration of link $0 \rightarrow 1$ in Figure 5a. Such paths toward Node 0 are shown for reconfiguration of link $1 \rightarrow 0$ in Figure 5b.

First, we evaluate a simple ring topology of 9 routers. The paths from every node to Node 1 are depicted in Figure 5a using black arrows. We measure the micro-loops duration during OSPF convergence after re-configuring the metric of link $(0, 1)$.



(a) Metric change.



(b) Link change.

Fig. 6: Duration of each micro-loop detected during OSPF re-convergence in the 9-routers ring topology after link $(0, 1)$ metric re-configuration.

a) Metric change: We first consider the case where the metric is modified only in direction $0 \rightarrow 1$. For this experiment, the median convergence duration over ten runs is 27.211 ms. Figure 5a shows the post-convergence paths as well as the potentially introduced micro-loops in red. Figure 6a shows the duration of the observed loops after the metric reconfiguration of link $0 \rightarrow 1$. It illustrates that every micro-loop predicted in Figure 5a happens in real-world executions. Loop $(0, 8)$, which is the nearest of the reconfigured link, is the longest of the detected loops. This is explained by the operations performed by Node 0. After the metric reconfiguration, it updates its RIB and originates a new LSA reflecting the update. Then, it recomputes its best routes and updates its FIB accordingly. Finally, it floods the newly originated LSA through the topology. Nodes 8, 7 and 6 require fewer operations. Upon new LSA reception, they inject it in their LSDB and then immediately flood it. Afterwards, they recompute their best routes and update their FIBs. The key element is that nodes neighboring the reconfiguration first update their FIB then flood the update while distant nodes flood the update before updating their FIB. Loop $(0, 8)$ begins after the FIB update of Node 0 and ends after the FIB update of Node 8. Both events are delayed by the link latency and internal OSPF operations, in particular by the duration of the FIB update which directly depends on the number of routes to update. We count a total of 10 FIB updates within the complete topology during the re-convergence phase. They are divided as follows; four for Node 0, three for Node 8, two for Node 7 and one for Node 6. As we observe in Figure 6a, the most important factor of a micro-loop is the cumulated link latency between the nodes defining the loop, here 5 ms. Hence, high latency links such as transatlantic ones, may have a dramatic impact on the duration of micro-loops. The experienced latency also validates our measurement methodology as the median duration of loops $(7, 8)$ and $(6, 7)$ are close to the link delay in our setup. Over the ten runs it lies around 5 ms.

b) Link change: We now consider the case where the link metric is updated in both directions. The second end of the link is reconfigured between 3 ms and 4 ms after the first end. This setup, referred to as *link change*, may emulate a link failure with active data-plane protection, as traffic on the link is not interrupted while the link is being removed from the topology. The median convergence duration over ten runs is 28.892 ms, which is a bit longer than in the metric change case as expected. Figure 6b shows the duration of each micro-loop predicted in Figures 5a and 5b. Similarly to the metric change case, the longest loops $(0, 8)$ and $(6, 7)$ are directly adjacent to the updated link. Their median duration is around 6 ms while it is 5 ms for distant loops. The median duration for loops $(0, 8)$, $(6, 7)$ and $(7, 8)$ is very similar in both the link change (Figure 6b) and metric change (Figure 6a) cases. We observe 20 FIB updates. Since the topology is perfectly symmetric, it is expected that both nodes neighboring the update trigger the same amount on FIB updates in the network.

V. Evaluating oFIQUIC

The quite small variance in micro-loops duration shown in Figure 6 indicates that our emulation framework is precise enough to consistently measure short-lived events triggered by routing protocols in a network. Showing every detected micro-loop for each evaluated topology would be cumbersome. Hence, in the remaining of this document, we only outline the duration of the loop that is the longest in the largest number of runs. We now leverage the DUNE framework to compare the behavior of our oFIB implementation with OSPF on simple synthetic (Section V-A, Section V-B) and a real-world (Section V-C) network topologies.

A. Evaluating oFIQUIC on a simple ring

We evaluate our oFIQUIC prototype on the 9-nodes ring (Figure 5) where Nodes 0 and 1 are scheduling nodes. Our oFIQUIC prototype successfully suppresses the micro-loops triggered by OSPF. In Figure 5a, Nodes 6, 7 and 8 are distant nodes explicitly contacted via QUIC tunnels by scheduling Node 0. Due to the structure of the topology, the total and partial ordering methods produce the same ordered list of distant nodes. Hence, both ordering methods should produce similar timing results.

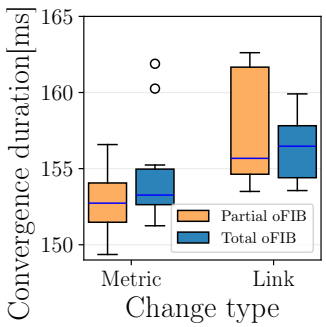


Fig. 7: Convergence duration for partial and total ordering of oFIQUIC on the 9-routers ring. Both metric and link change are shown.

Figure 7 shows the convergence duration of oFIQUIC for both partial and total ordering. In the metric change case, the median convergence duration on ten runs is 152.733 ms for partial ordering against 153.263 ms for total ordering. In the link change case, the median convergence duration on ten runs is 155.676 ms for partial ordering against 156.474 ms for total ordering. This difference is considered negligible as expected. The duration difference between metric and link change cases is explained by the varying delays between convergence start on both scheduling nodes.

Figure 8 shows the main temporal contributions for partial oFIQUIC ordering during link $(0, 1)$ change in the 9-routers ring. We first consider the contributions for scheduling Node 0. Upon link reconfiguration, Node 0 computes distant nodes ordering $[6, 7, 8]$, as defined by the micro-loops predicted in Figure 5a. It asynchronously opens a QUIC session with each of them. The nearest the distant node is from the scheduling node, the quicker the QUIC session is established. Node 6,

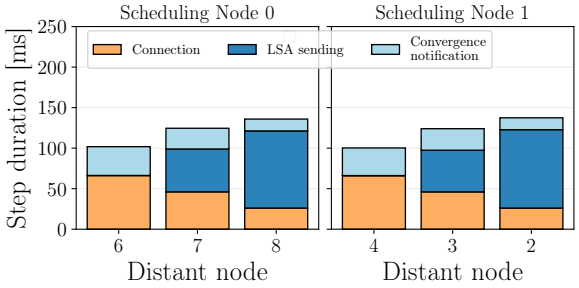
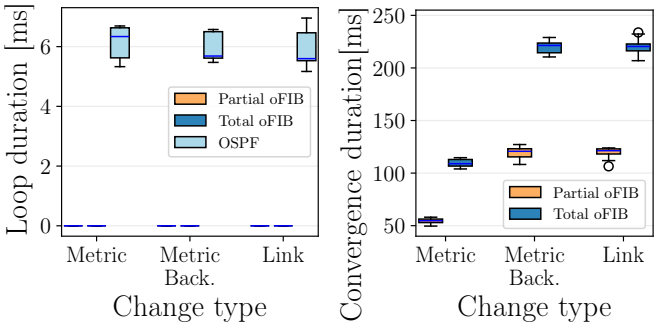


Fig. 8: Temporal contribution of every oFIQUIC step during metric re-configuration of link $(0, 1)$ with partial ordering.

which is the first to be updated, is the last whose session is effectively established as it is the furthest. After sending its newly originated LSA, Node 0 waits for the convergence notification from the distant node. This step is the longest for distant Node 6 as it is the furthest. Again, the nearest the distant node is from the scheduling node, the quicker this step is accomplished. The median duration for updating Node 6 over ten runs is 101.826 ms. Median contributions for scheduling Node 1 are very similar as for Node 0 since the topology is perfectly symmetric.

B. Evaluating oFIQUIC on a simple 3×3 grid

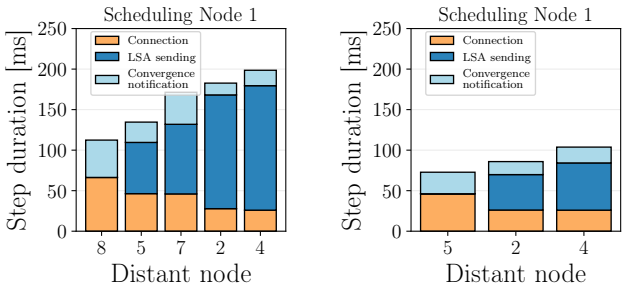
We evaluate our oFIQUIC prototype on a simple 3×3 grid topology depicted in Figure 2a.



(a) Most frequent longest loop. (b) Convergence duration.

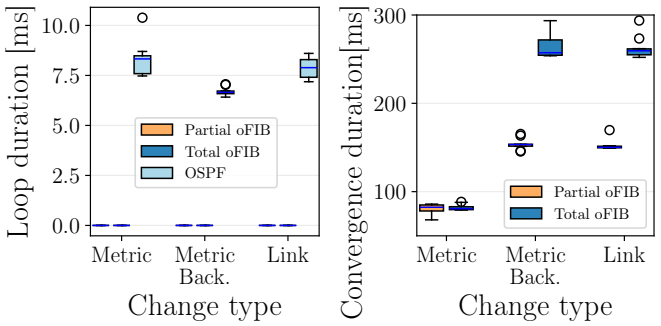
Fig. 9: Most frequent longest loop duration (9a) measured during metric re-configuration of link $(0, 1)$ and convergence duration (9b) of oFIQUIC on the 3×3 grid (Figure 2a).

Figure 9a shows the duration of the most frequent longest loop averaging around 6 ms during the metric update of link $(0, 1)$. We observe that oFIQUIC successfully suppresses all micro-loops. Figure 9b depicts the convergence duration of oFIQUIC. It validates that partial ordering can perform significantly better than total ordering, as seen in the convergence duration for link $(0, 1)$ metric re-configuration. In the case of metric change in the $1 \rightarrow 0$ direction (noted *Metric. Back.* in Figure 9b), the median duration is 221.381 ms for total ordering and 120.736 ms for partial ordering. This difference is explained by the oFIQUIC contributions depicted in Figure 10. Total ordering forces Node 1 to contact five distant nodes (Figure 10a), with Node 8 being the furthest at



(a) Total ordering (b) Partial ordering

Fig. 10: oFIQUIC contributions during metric re-configuration of link $1 \rightarrow 0$



(a) Most frequent loop duration. (b) Convergence duration.

Fig. 12: Performance evaluation upon metric reconfiguration of link between Denver and Kansas City in Abilene.

three hops. Establishing a QUIC session and waiting for an explicit convergence notification takes 112.399 ms, which is quite costly knowing that this node does not trigger transient events while receiving the new LSA. On the other hand with partial ordering, the scheduling node contacts three distant nodes (Figure 10b) whose furthest Node 5 is two hops away. Successfully triggering re-convergence on this distant node takes on average 72.773 ms. Contacting mainly nearby distant nodes allows nearly halving the convergence time in this situation.

C. Evaluating oFIB on Abilene

We evaluate our oFIQUIC prototype on the Abilene topology[2] depicted in Figure 11.

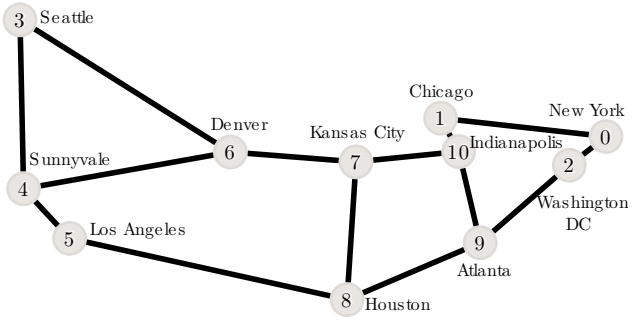


Fig. 11: Abilene network.

Figure 12a shows the duration of the most frequent longest micro-loop detected upon the metric reconfiguration of the link between Denver and Kansas City. It averages around 7 ms for OSPF while oFIQUIC successfully suppresses all micro-loops. Convergence duration for this update is depicted in Figure 12b. It mainly depends on the number of distant nodes needed to be contacted by scheduling nodes. This is clearly visible for the metric change of link Denver $\rightarrow$ Kansas City, noted *Metric.* on Figure 12b. By looking at oFIQUIC contributions of scheduling node Denver, it contacts a single distant node in partial ordering (Figure 14) against two in total ordering (Figure 13). However, in the case of metric change of the link Kansas City $\rightarrow$ Denver (noted *Metric. Back.* in Figure 12b), the scheduling node contacts three distant nodes in partial ordering against six in total ordering. This contribution is also the most significant in the link change case.

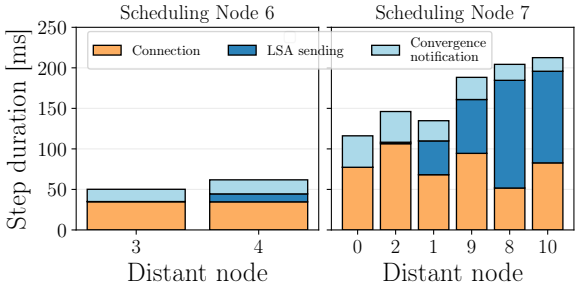


Fig. 13: oFIQUIC contribution with total ordering upon metric reconfiguration of the link between Denver (Scheduling Node 6) and Kansas City (Scheduling Node 7) in Abilene.

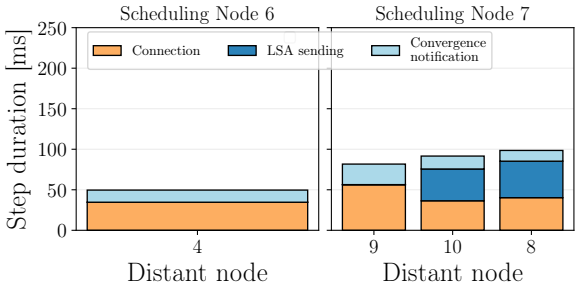


Fig. 14: oFIQUIC contribution with partial ordering upon metric reconfiguration of the link between Denver (Scheduling Node 6) and Kansas City (Scheduling Node 7) in Abilene.

D. Performance considerations of oFIQUIC

With this evaluation, we showed that oFIQUIC successfully suppresses micro-loops during OSPF convergence. We observe that the main temporal cost of oFIQUIC is the establishment of a QUIC session with the furthest distant node(s). This step, which depends on the RTT between the scheduling node and the distant node(s), delays the global ordered convergence. The

[2]https://web.archive.org/web/20120324103518/http://www.internet2.edu/pubs/200502-IS-AN.pdf

second temporal cost is the time waited before receiving the convergence notification. It also highly depends on the link delays.

VI. Conclusion

Link-state routing protocols such as OSPF rely on flooding. This ensures that all routers eventually receive all topology changes. Unfortunately, link-state routers update their forwarding tables asynchronously after each topology change which can result in transient forwarding loops.

In this paper, we propose oFIQUIC an extension to the OSPF protocol which brings two innovations. First, oFIQUIC uses the QUIC protocol instead of OSPF's specialized transport layer to exchange link state information between neighboring routers. This prevents various packet injection attacks and allows OSPF to benefit from QUIC's finely tuned reliability mechanisms. Second, oFIQUIC supports loop-free convergence after topology changes. When a topology change occurs, a scheduling router computes a loop-free ordering to update the forwarding table of all routers affected by the change. It then contacts each affected router using a secure QUIC session such that the forwarding tables are updated in an order that prevents loops. We implement oFIQUIC inside the BIRD routing daemon. Our evaluation in the `DUNE` evaluation framework shows that oFIQUIC prevents loops while ensuring a fast convergence. **Artefacts of the paper are available at** https://github.com/nrybowski/ofiquic-artefacts. Our further work will be to explore in more details how OSPF and other routing protocols could benefit from a secure transport protocol such as QUIC.

References

[1] M. Piraux *et al.*, "Revealing the evolution of a cloud provider through its network weather map," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 298–304.

[2] R. Jacob *et al.*, "Does rate adaptation at daily timescales make sense?" in *HotCarbon'23*. New York, NY, USA: ACM, 2023.

[3] A. Feldmann *et al.*, "The lockdown effect: Implications of the covid-19 pandemic on internet traffic," in *IMC'20*. New York, NY, USA: ACM, 2020, p. 1–18.

[4] J. Moy, "OSPF Version 2," RFC 2328, Apr. 1998.

[5] H. Gredler *et al.*, *The complete IS-IS routing protocol*. Springer Science & Business Media, 2005.

[6] P. Merindol *et al.*, "A fine-grained multi-source measurement platform correlating routing transitions with packet losses," *Computer Communications*, vol. 129, pp. 166–183, 2018.

[7] P. Francois *et al.*, "Achieving sub-second IGP convergence in large IP networks," *ACM SIGCOMM CCR*, vol. 35, no. 3, pp. 35–44, 2005.

[8] M. Shand *et al.*, "IP Fast Reroute Framework," RFC 5714, Jan. 2010.

[9] M. Chiesa *et al.*, "A survey of fast-recovery mechanisms in packet-switched networks," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1253–1301, 2021.

[10] A. Atlas *et al.*, "Basic Specification for IP Fast Reroute: Loop-Free Alternates," RFC 5286, Sep. 2008.

[11] C. Filsfils *et al.*, "Loop-Free Alternate (LFA) Applicability in Service Provider (SP) Networks," RFC 6571, Jun. 2012.

[12] S. Bryant *et al.*, "Remote Loop-Free Alternate (LFA) Fast Reroute (FRR)," RFC 7490, Apr. 2015.

[13] A. Bashandy *et al.*, "Topology Independent Fast Reroute using Segment Routing," IETF, draft-ietf-rtgwg-segment-routing-ti-lfa-13, Jan. 2024.

[14] P. Francois *et al.*, "Disruption free topology reconfiguration in OSPF networks," in *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 89–97.

[15] ——, "Avoiding transient loops during the convergence of link-state routing protocols," *IEEE/ACM Transactions On Networking*, vol. 15, no. 6, pp. 1280–1292, 2007.

[16] M. Shand *et al.*, "Framework for Loop-Free Convergence Using the Ordered Forwarding Information Base (oFIB) Approach," RFC 6976, Jul. 2013.

[17] S. Litkowski *et al.*, "Micro-loop Prevention by Introducing a Local Convergence Delay," RFC 8333, Mar. 2018.

[18] M. Thomson *et al.*, "Using TLS to Secure QUIC," RFC 9001, May 2021.

[19] A. Langley *et al.*, "The QUIC transport protocol: Design and internet-scale deployment," in *Proceedings of the conference of ACM SIGCOMM*, 2017, pp. 183–196.

[20] J. Iyengar *et al.*, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021.

[21] C. Huitema *et al.*, "DNS over Dedicated QUIC Connections," RFC 9250, May 2022.

[22] M. Trevisan *et al.*, "Measuring the performance of icloud private relay," in *PAM*. Springer, 2023, pp. 3–17.

[23] P. Sattler *et al.*, "Towards a tectonic traffic shift? investigating apple's new relay network," in *Proceedings of the 22nd ACM IMC*, 2022, pp. 449–457.

[24] L. Curley *et al.*, "Media over QUIC Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-moq-transport-01, Oct. 2023, work in Progress.

[25] T. Wirtgen *et al.*, "Routing over quic: Bringing transport innovations to routing protocols," *arXiv preprint arXiv:2304.02992*, 2023.

[26] D. Watson *et al.*, "Experiences with monitoring OSPF on a regional service provider network," in *23rd ICDCS proceedings*. IEEE, 2003, pp. 204–213.

[27] S. Lee *et al.*, "To automate or not to automate: on the complexity of network configuration," in *IEEE ICC*. IEEE, 2008, pp. 5726–5731.

[28] R. Jacob *et al.*, "The internet of tomorrow must sleep more and grow old," *ACM SIGENERGY*, vol. 3, no. 3, pp. 27–32, 2023.

[29] B. Fortz *et al.*, "Traffic engineering with traditional IP routing protocols," *IEEE communications Magazine*, vol. 40, no. 10, pp. 118–124, 2002.

[30] A. Sridharan *et al.*, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions On Networking*, vol. 13, no. 2, pp. 234–247, 2005.

[31] B. Fortz *et al.*, "Optimizing ospf/is-is weights in a changing world," *IEEE JSAC*, vol. 20, no. 4, pp. 756–767, 2002.

[32] A. Markopoulou *et al.*, "Characterization of failures in an operational IP backbone network," *IEEE/ACM ToN*, vol. 16, no. 4, pp. 749–762, 2008.

[33] F. Clad *et al.*, "Graceful convergence in link-state IP networks: A lightweight algorithm ensuring minimal operational impact," *IEEE/ACM ToN*, vol. 22, no. 1, pp. 300–312, 2013.

[34] ——, "Computing minimal update sequences for graceful router-wide reconfigurations," *IEEE/ACM ToN*, vol. 23, no. 5, pp. 1373–1386, 2014.

[35] K.-T. Foerster *et al.*, "Survey of consistent software-defined network updates," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1435–1461, 2018.

[36] M. Shand *et al.*, "A Framework for Loop-Free Convergence," RFC 5715, Jan. 2010.

[37] M. Fanto *et al.*, "OSPFv2 HMAC-SHA Cryptographic Authentication," RFC 5709, Oct. 2009.

[38] M. Gupta *et al.*, "Authentication/Confidentiality for OSPFv3," RFC 4552, Jun. 2006.

[39] D. Ferguson *et al.*, "OSPF for IPv6," RFC 5340, Jul. 2008.

[40] J. Iyengar *et al.*, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021.

[41] T. R. Henderson *et al.*, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[42] A. Varga *et al.*, "An overview of the omnet++ simulation environment," in *1st International ICST SIMUTOOLS*, 2010.

[43] B. Lantz *et al.*, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[44] R. Jansen *et al.*, "Co-opting linux processes for {High-Performance} network simulation," in *USENIX ATC'22*, 2022, pp. 327–350.

[45] H. Tazaki *et al.*, "Direct code execution: Revisiting library os architecture for reproducible network experiments," in *Proceedings of ACM CoNEXT'13*, 2013, pp. 217–228.

[46] H. Li *et al.*, "Simbricks: end-to-end network system evaluation with modular simulation," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 380–396.

[47] N. Rybowski *et al.*, "Evaluating ospf convergence with ns-3 dce," in *Proceedings of the 2022 Workshop on ns-3*, 2022, pp. 120–126.