Future Generation Computer Systems 140 (2023) 104–116

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

EdgeVPN: Self-organizing layer-2 virtual edge networks

Kensworth Subratie, Saumitra Aditya, Renato J. Figueiredo *

Advanced Computing and Information Systems (ACIS) Lab, Electrical and Computer Engineering (ECE) Department, University of Florida, 968 Center Drive, Gainesville, 32611, FL, USA

ARTICLE INFO

Article history:
Received 12 May 2022
Received in revised form 27 August 2022
Accepted 6 October 2022
Available online 14 October 2022

MSC:
68M10
68M14

Keywords:
Edge computing
Fog computing
Virtualization
Overlay networks
Peer-to-peer
Software-defined networks

ABSTRACT

The advent of virtualization and cloud computing has fundamentally changed how distributed applications and services are deployed and managed. With the proliferation of IoT and mobile devices, virtualized systems akin to those offered by cloud providers are increasingly needed geographically near the network's edge to perform processing tasks in proximity to the data sources and sinks. Latency-sensitive, bandwidth-intensive applications can be decomposed into workflows that leverage resources at the edge — a model referred to as fog computing. Not only is performance important, but a trustworthy network is fundamental to guaranteeing privacy and integrity at the network layer. This paper describes Bounded Flood, a novel technique that enables virtual private Ethernet networks that span edge and cloud resources — including those constrained by NAT and firewall middleboxes. Bounded Flood builds upon a scalable structured peer-to-peer overlay, and is novel in how it integrates overlay tunnels with SDN software switches to create a virtual network with dynamic membership — supporting unmodified Ethernet/IP stacks to facilitate the deployment of edge applications. Bounded Flood has been implemented as the core of the EdgeVPN open-source virtual private network software system for edge computing. Experiments with the software demonstrate its functionality and scalability — one of which includes Kubernetes with Flannel across Raspberry Pi 4 edge devices behind different NATs.

1. Introduction

We are on the cusp of a new era of smart devices, the Internet of Things, and smart spaces [1]. These ideas herald a degree of social and technological change that will have profound impacts on the way humans interact with technology, including cyber–physical systems that underscore the sense-analyze-actuate model [2], and pervasive computing [3,4]. The generalized sense-analyze-actuate model is widely applicable to various aspects of everyday life, including farming, autonomous vehicles, commerce, and healthcare.

Emerging prototypes of these applications already exhibit requirements that are difficult to meet using existing cloud computing models [5]. Alternative approaches being investigated include edge, multi-access edge, cloudlet, mist, and fog computing [6]. These leverage lightweight data centers distributed across the network's edge as processing nodes to bring compute and short-term storage closer to the data sources and sinks. This eliminates the latency and throughput penalties incurred from moving data across large geographic distances and through high contention, bandwidth-limited links. However, it introduces an operation and management problem: it is necessary to interconnect all widely distributed components to create a virtualized computing environment. Unfortunately, software and methodologies designed for the data center are typically poorly suited for fog computing operations along the Internet's edge due to constraints of the Internet Protocol (IP).

The IPv4 protocol was never designed for the scale and complexity of the modern Internet, and IPv6 has not yet seen wide deployment, leaving most of the Internet still operating on IPv4 [7,8]. A lack of public IPv4 addresses has been partially mitigated by Network Address Translators (NAT); however, these hinder peer communication as NATed devices do not have an address that is routable over the public Internet. Furthermore, the lack of built-in security and privacy in the IPv4 protocol means application-level or transport-level mechanisms must be employed. While within cloud computing data centers nodes can communicate without the presence of NATs, edge computing applications that require devices to communicate across different private networks must deal with NAT traversal.

Network virtualization stands at a unique vantage point to address these challenges. While existing Virtual Private Networks (VPNs) can mitigate hurdles such as endpoint addressing and secure communication, current VPN systems that use cloud-hosted routers run counter to the key edge computing goal of processing near data. In particular, as future IoT applications are

* Corresponding author.
E-mail address: renatof@ufl.edu (R.J. Figueiredo).

https://doi.org/10.1016/j.future.2022.10.007

envisioned to leverage geo-distributed, multi-edge clusters [9], there is a need for direct edge-to-edge communication among geographically-close edge data centers to support flexible orchestration. However, traditional tunneling-based VPNs lack support for NAT traversal, while MPLS-based VPNs require configuration of backbone equipment [10]. This need motivates a scalable system that supports dynamic membership across multiple sites, virtualizes addressable endpoints to support existing software, natively supports peer-to-peer communication behind NATs, and provides secure communications.

Edge networks [11], as used in the recent IoT context, refer to the local networks that connect IoT devices (such as Wi-Fi access points and gateways) and exclude the IoT devices themselves. While Software-Defined Networking (SDN) provides a foundation, current SDN-based approaches employed in data center network architecture and administration [12,13] are not well suited for building and managing dynamic edge networks connecting multiple providers. Challenges arise from: (1) the geographic distances over which equipment must be deployed and the effort to physically access these locations; (2) the heterogeneous mix of components and configuration; (3) networks that must span multiple administrative control domains; and (4) dynamic membership than can often be short-lived. New approaches are necessary to address these issues, leading to the observation that virtualized networks are positioned to play an important role in the orchestration and communication among geographically distributed containerized applications [14–18].

The main contribution of this paper is the design, implementation, and experimental evaluation of a system that is novel in how it supports structured peer-to-peer (P2P) control and data plane for software-defined virtual private networking (VPN). While previous work has described tradeoffs and conceptual choices leading to the P2P VPN design [19], this paper presents for the first time a detailed discussion of the underlying mechanisms and protocols supporting scalable layer-2 Ethernet broadcast and unicast and a comprehensive experimental evaluation of the performance of an open-source implementation that supports unmodified middleware and applications, specifically for edge and fog computing. The Bounded Flood (BF) technique described in this paper enables scalable VPNs with an overlay topology that supports heterogeneous node capabilities and configuration while providing private edge-to-edge, peer-to-peer (P2P) tunnels among members behind NATs. Furthermore, BF supports multiple networks with dynamic membership and decentralized, self-configuring forwarding rules on standard SDN switches for Ethernet broadcast and unicast.

This paper presents the novel design of Bounded Flood and describes its implementation in the open-source EdgeVPN software [20]. The implementation is used to qualitatively validate and quantitatively assess the functional requirements of the system, specified across the following seven broad categories:

(F1) Decentralization: (a) Each node acts independently of the others with respect to network processing, such that (b) layer-2 switching is fully distributed.

(F2) Scalability: (a) BF scales as network size increases, and (b) there are bounds on switching hops and maintenance cost (adding/removing a node) relative to network size.

(F3) Availability: (a) After topology changes, a path of bounded length between two nodes must eventually be found if one exists. (b) The protocol must support arbitrary arrival and departure of nodes, where (c) link creation/trimming does not cause overlay-wide communication interruption. (d) All overlay links are available for use and nodes can create and utilize direct links as needed. (e) Existing unicast sessions should proceed unaffected to the extent allowed by underlying physical connectivity.

(F4) Fault Tolerance: (a) The system must tolerate a known degree of failure within the overlay without creating partitioned networks. (b) Beyond this limit, it must remain operable within individual partitions.

(F5) Resilience: (a) The system must autonomously repair a partitioned overlay and resume overlay-wide functionality when inter-partition links are restored.

(F6) Flexibility: (a) The system must provide multiple independent Ethernet broadcast domains with no forwarding loops. (b) Applications must have flexibility in how they assign and relocate layer-3 virtual IP addresses. (c) Functional parameters must be configurable for operational tuning, and nodes can be configured independently.

(F7) Simplicity: (a) The system design must be practical to implement and (b) require simple operational configuration.

The rest of this paper is organized as follows. Section 2 introduces technologies that are foundational to this work and surveys related works. Section 3 details the design of Bounded Flood and the implementation in EdgeVPN, while Section 4 describes the basis for its functional and qualitative evaluation. Section 5 presents the experimental findings and interprets the results. Section 6 considers future work, and presents concluding remarks.

2. Background

While there is an extensive literature and implemented systems on overlay networks, SDNs, and virtual networks, the approach described in this paper is novel. It is the first system to provide self-assembling virtual private networks built on a P2P overlay with scalable, software-defined and self-organizing flows for Ethernet broadcast/unicast across decentralized SDN switches, delivering virtual networking across devices in private networks constrained by NATs.

2.1. Ethernet, SDN, and network virtualization

The link layer (layer 2), e.g., Ethernet [21], is concerned with the point-to-point transmission of frames between devices connected by a link. Multiple devices can be connected to a network bridge and a switched network fabric to enable communication between any pair of endpoint devices. Hosted devices or leaf devices are differentiated from bridges or switches in that they perform no switching functionality. Each leaf or bridge device that participates in the same Ethernet namespace is also in the same broadcast domain. Ethernet has no support for loops, and specialized procedures must be implemented to protect against layer 2 broadcast storms. One approach is to design a network architecture that is loop-free, such as a hierarchical topology. In the absence of a loop-free topology, techniques must implement protocols to avoid loops that may occur naturally in a fabric. A simple and widely used approach that can be implemented in the data plane is the Spanning Tree Protocol (STP) [22,23]. In STP, all the bridges in the domain agree on a root bridge that becomes the central coordinator in identifying and disabling redundant links until a spanning tree remains. This approach, however, has several drawbacks: domain-wide communication is interrupted each time a bridge is added or removed as the spanning tree must be recalculated; multiple links are disabled and left idle; contention for the existing links is increased, and the overall throughput of the fabric is reduced. Furthermore, these drawbacks are exacerbated in dynamic, distributed environments that are found in fog and edge computing.

There are other approaches [24] that implement software-based switching based on expectations of the topology. Critical to the feasibility of such approaches is Software Defined Networking (SDN), which separates the switch into two components, the control and data planes. The data plane is concerned with efficiently moving network frames across links, while the control plane is

implemented in software using APIs, libraries, and frameworks such as OpenFlow [12], Ryu [25], and Open vSwitch. Software-Defined WANs (SD-WAN [26]) is an approach to describe and control wide-area networks with monitoring, traffic engineering, and QoS guarantees. While our work is related in that it also uses SDN switches for wide-area networks, a key difference is that in these systems there is a (logically) *centralized* SDN controller typically under a *single administrative domain*, while in EdgeVPN, *SDN controllers are decentralized along with the data plane*, allowing the system to scale as nodes join the overlay and leading to no single point of failure in the control plane.

Virtualizing a communication network by protocol tunneling is widely used as it is both powerful and flexible. Network Function Virtualization (NFV) [27] and SDN enable the means by which networks can be provisioned and operated entirely in software. Virtual networks are now widely used to connect tenants within and across data centers — in particular, when container orchestration frameworks such as Kubernetes integrate virtual networks to create a fully virtualized infrastructure. The approach proposed in this paper aims to support containerized platforms to be deployed, unmodified, across multiple edge and cloud providers, and is novel in how it provides seamless connectivity at the virtual network's Ethernet layer despite connectivity constraints due to the asymmetry introduced by NATs/firewalls at the physical Internet layer. This is a key difference with respect to traditional tunneling-based VPNs that lack support for NAT traversal between sites [10].

2.2. Overlay and P2P networks

Overlay networks are application-level implementations of value-added network services built on existing underlying communication infrastructure. They provide functionalities such as resilient routing [28], distributed data structures [29], and multicast [30]. Exploiting the properties of how nodes in an overlay are arranged and interrelated can yield beneficial results. This has been shown in the study of distributed hash tables (DHT) and structured P2P systems [29,31–33]. Structured P2P overlays such as Chord [29], Kademlia [34] and Symphony [33] have been shown to be resilient, scalable, and to work well for large and small networks alike — even when participants independently arrive and depart at arbitrary intervals. In this work, we leverage structured overlays for a different purpose, and in a novel way — as a fabric for virtual networking. In the virtualized overlay network approach of this paper, SDN switches are the nodes (vertices) in the overlay, and Internet tunnels are the links (edges) connecting among nodes. Overlays can extend from cloud to edge, enabling existing methodologies and utilities to be used for orchestrating IoT containerized applications and services.

While overlay networks have been used through interfaces that include virtual network devices at layer-2 or layer-3 [35,36], IP forwarding proxies [28], "convergence" layer below the transport stack [37], or "smart sockets" [38] at the application layer, this paper presents a novel approach where overlay links are initiated and terminated at switch ports of a distributed fabric of SDN switches organized as a P2P overlay and where endpoints may have private, non-routable Internet addresses.

2.3. Virtualized infrastructure

The Container Network Interface (CNI) for Kubernetes is a set of specifications and associated libraries for developing plugins that configure network interfaces in Linux containers. The CNI specification covers the network connectivity of containers and the deallocation of resources when a container is terminated; it is supported by Flannel [16], Calico [17], WeaveNet [18], Romana [39], and NSX [40]. As outlined below, CNI solutions vary in sophistication and the number of supported virtualization levels. However, they all assume that cluster nodes are routable — an assumption that breaks when edge computing nodes are NATed.

Flannel limits the scope of its functionality to a layer 3 IPv4 network between multiple nodes in a cluster, giving an IP subnet to each host for use with container runtimes. Project Calico is a network security solution that virtualizes the IP layer and attempts to avoid tunneling when workload endpoints are addressable. Weaveworks WeaveNet uses VxLAN encapsulation to virtualize a layer 2 network and forwards traffic between nodes in a mesh network. Weave peers communicate their knowledge of the topology so that all peers learn about the entire topology. Communication between peers occurs over TCP links using a spanning-tree-based broadcast with neighbor gossip.

Romana Cloud Native Networks uses layer 3 network techniques to build micro-segmented, cloud-native networks without a virtual network overlay. Micro-segmentation is a method of creating secure zones in data centers and cloud deployments by logically dividing the data center into distinct security segments down to the individual workload level, and then defining security controls and delivering services for each unique segment.

VMware NSX also provides network and security virtualization using micro-segmentation. Additionally, NSX provides layers 2–4 (including switching, routing, load balancing, micro-segmentation, and distributed firewalling) networking and security virtualization platform by running these network layers' stack in software, decoupled from underlying physical hardware.

Midonet [41] provides virtualized layers 2–4 networking with distributed layer 2 switching and layer 3 routing, in addition to distributed load balancing and firewall services. Midonet uses VxLAN for its layer 2 networking and supports integration with VxGW (VXLAN Gateway) and hardware VTEP (VXLAN Tunnel End Point) as specified in [42].

IBM Dove [43] (and its successor SDN VE) is an SDN-based network virtualization solution built around the concept of centrally controlled, platform terminated overlays and allows the creation of multiple, isolated, and dynamic virtual networks over shared physical infrastructure. It implements a proprietary NFV switch as the datapath, which virtualizes both layers 2 and 3. The system is managed from a centralized console that configures each virtual network, and controls and disseminates policies to the virtual switches. The administrator uses an intent-based modeling abstraction for specifying the network as a policy governed service and thus expressing the functionality of the desired virtual network.

Google Andromeda [44] is the network virtualization environment for the Google Cloud Platform. It is a multi-component, distributed system, using a hierarchical architecture that provides layers 2–4 virtualization with features such as switching, routing, monitoring, and firewall protection. To support throughput and latency similar to what is available from the underlying hardware, Andromeda uses a data plane that combines a set of user-space packet processing paths to handle specialized workloads. The Andromeda control plane is designed around a global hierarchy, where every cluster runs a separate control stack for isolation. It maintains information about where every VM in the network currently runs, and through a hierarchy of controllers, selected subsets of the controller state are installed at individual servers.

Zerotier [45] is developed as a global-scale VPN with SDN management capabilities. It is functionally similar to EdgeVPN but with distinct technical differences. Zerotier's architecture implements two separate software layers, VL1 and VL2. The VL1 layer is responsible for virtual link management, maintenance of the DNS-like topology, and packet transport. A key difference from

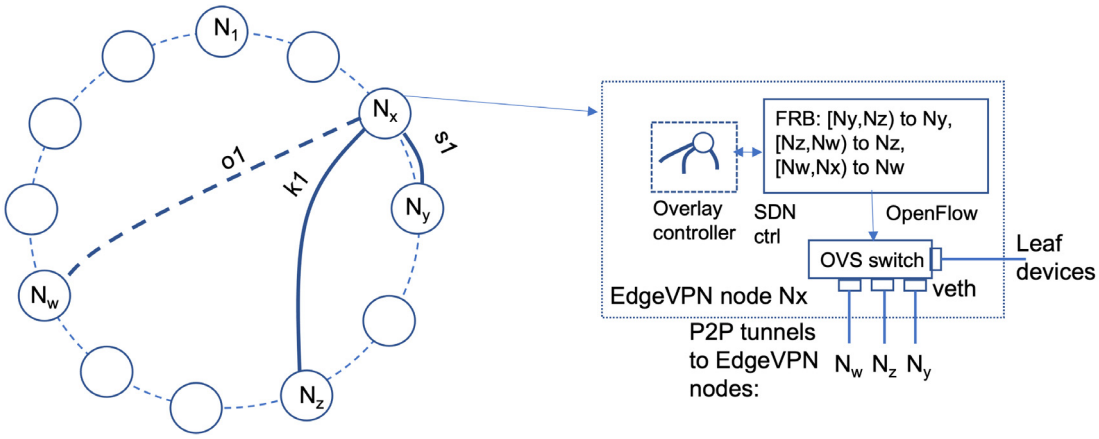

Fig. 1. System Overview. The EdgeVPN overlay consists of nodes identified by unique IDs and organized in a structured P2P topology based on Symphony. Node N_x has a successor link $s1$ to node N_y, and a long-distance link $k1$ to node N_z; an on-demand link $o1$ is being established in this example, in response to sustained communication between N_x and N_z. The overlay controller maintains a local view of the node's links – each is a P2P tunnel that terminates to a virtual Ethernet device – and is connected to a local Open vSwitch (OVS) switch port. The SDN controller coordinates with the overlay controller to craft Bounded Flood FRB messages upon broadcast requests, which are encapsulated, encrypted, and sent over P2P tunnels.

EdgeVPN is topology: ZeroTier uses twelve globally maintained root servers in the V1 layer(with support for user-added sub-root servers), while in EdgeVPN there are no roots but rather a structured P2P ring. Each root node tracks its sub-nodes and uses this information for node discovery and establishing peer connections. Requests are propagated up the hierarchy from an endpoint to the nearest root node until the target is known and can be forwarded along the appropriate path. Root nodes use this information to provide endpoint hints that can be used for NAT traversal and building peer links. When NAT traversal is impossible, root nodes can act as relay hosts. V1 connections are cryptographically secured with end-to-end encryption, and each node is uniquely identified on VL1 by a 40-bit (10 hex digit) ZeroTier address. Additionally, ZeroTier virtual network is managed by a single network controller, whereas in EdgeVPN the SDN control is fully decentralized. Finally, a third major difference is that Zerotier implements its own SSL-like capabilities for secure transmissions and its own NAT traversal, while EdgeVPN uses standards (SSL, STUN, and TURN).

SoftEther [46] is a multi-protocol VPN that tunnels Ethernet over HTTPS (SoftEther protocol), L2TP over IPSec, PPP over HTTPS (Microsoft SSTP), and IP over UDP (OpenVPN). SoftEther implements roles for VPN clients and bridges, where clients connect to bridges, and bridges can be connected to create ad-hoc structures. The UDP mode of SoftEther VPN supports NAT traversal. SoftEther provides compatibility with many traditional VPNs but does not support automatic link management, dynamic membership, or network fabric (network topology with cycles).

In summary, while several virtualization solutions exist (e.g. Calico, Flannel), many of them only work if all nodes in a cluster are routable, i.e. in the same network address space (such as within a cloud data center), or across the Internet (if all nodes have public IP addresses). Additionally, these related solutions do not support dynamic membership, and they do not apply to the environments that this paper focuses on – multi-edge deployments – and therefore, we do not perform a direct comparison in the evaluations. Nonetheless, we demonstrate that, while CNIs such as Flannel do not work natively when nodes are NATed, they *do work, unmodified*, on top of EdgeVPN (Section 4). In other words, compared to these related works, the techniques described in this paper are different in a fundamental way to cater to edge computing use cases. Bounded Flood works in a decentralized fashion using a structured P2P overlay topology coupled to a decentralized set of SDN controllers, and as a system, EdgeVPN is unique in how it integrates decentralized control and a datapath that recovers connectivity across NATed devices using standard NAT traversal and security protocols (STUN, TURN, SSL), thereby enabling existing software (including Kubernetes with Flannel) to work unmodified both edge-to-cloud and edge-to-edge.

3. Design

The goal of Bounded Flood is to deliver scalable layer 2 forwarding for dynamic edge and cloud network environments where the peer nodes act as software-defined bridges. Bounded Flood has been implemented in a virtual network software package, which integrates a structured P2P topology and a decentralized SDN-enabled layer-2 switching fabric. Each BF node runs two major modules (Fig. 1): (1) the Overlay Controller is solely focused on overlay creation and maintenance, while (2) the SDN Controller programs the corresponding switching rules. Each node or peer runs the same software with the same functional behavior, and independently maintains its instances of the Overlay and SDN controllers. While there are no centralized components for overlay management and SDN-programmed switching, it uses a messaging service (XMPP [47] for peer authentication and messaging, and STUN/TURN [48,49] servers for endpoint discovery and tunnel bootstrapping.

3.1. Overlay controller

The Overlay Controller's topology module builds a Symphony (1-D Kleinberg routable small-world network [50]) structured overlay in a P2P fashion, where each link is an Internet tunnel supporting NAT traversal (Fig. 1). The topology defines three types of links: successor, long-distance, and on-demand. Successors are arranged on the ring's perimeter, in increasing order by their unique ID, and each node is configured to link to its s closest successors. As each node creates its successor links (accommodating for wrap-around) the loop is closed and forms the ring. The ordered ring ensures a foundation for P2P messaging: nodes are routable using greedy forwarding based on overlay identifiers.

On-demand links are used elastically: they are created and destroyed as needed to establish a direct tunnel between peers based on observed demand. Using on-demand tunnels reduces the switching hops and removes the traffic burden on intermediate bridges that lie on the communication path. Long-distance

links are used as shortcuts across the ring and reduce the average path length between any two nodes. Bounded Flood implements shortcuts based on Symphony long-distance links [33]. In an overlay with n nodes, each node maintains $k \geq 1$ long-distance links and selects long-distance peers by drawing a random number x from the following probability distribution function (pdf):

$$p_{(x=n)} = e^{(log(n)*(rand(0,1)-1.0))} \quad (1)$$

The product $n * x$ specifies the clockwise distance, from the source node to the furthest peer. When creating a link, the closest node ID at a distance less than $n * x$ is selected.

There are two components integral to the functionality provided by the Overlay Controller's topology module. They are the Network Graph Builder, which creates a representation of the desired network state (from the local standpoint of the node), and the Network Orchestrator, which transitions the node from its current state to the desired one. As nodes join and leave the overlay (a process known as churn) the composition and size of the graph changes. Peers which were previously ideal candidates for links become less favorable, triggering the graph builder to reevaluate the desired link set.

The Network Graph Builder receives as its input the currently available link candidates, targeted amounts for each type of link, and the current local network graph (i.e. the peer's adjacency list). It selects new successor and long-distance nodes based on available candidates and replaces existing, less favorable ones. A long-distance peer is considered too close and is discarded if it falls within the distance given by:

$$D_c = \lfloor (n * e^{(-1*log(n))}) \rfloor \quad (2)$$

When configured to maintain k long-distance links per node, k candidates are selected as long-distance peers and k peers can select this node as their long-distance peer (i.e. $2k$ long-distance links per node). Nodes must be prepared to accept as many links as they create, or the resulting imbalance would cause cascading failures in link creation.

The Network Orchestrator uses the generated network graph to modify the node's adjacency list. The differences between the current and desired states provide the context to generate the new network state via update, remove, and add link operations. As each node acts independently, each Network Orchestrator effectively manages a star graph with the local node at the center of its adjacent peers. It is by the graph builder's careful selection of candidates at each peer that the distributed set of local topologies becomes the structured P2P ensemble. The Network Orchestrator reliably transitions the local network state by identifying the differences between the current realized state and the desired state. It removes deprecated links and initiates the creation of those that need to be added. It prioritizes creating successor links before removing existing ones to avoid partitioning the overlay's ring and to maintain routability. It negotiates with peers to create links, rejecting requests when link quantity thresholds are reached. It handles link interruption and bootstrapping failure, ensuring the overlay's network state is consistent and the node participation is correct.

3.2. SDN controller

The novel Bounded Flood SDN Controller developed in this work is a Ryu-based [25], OpenFlow-compliant [12] module that utilizes a specialized method for broadcast that exploits the structured P2P topology to discover acyclic network paths between peer nodes in a dynamic, decentralized way. Each node along a network path uses its local information to build its data plane forwarding database. Each forwarding record in the database is an OpenFlow flow rule which specifies the egress action when a specific ingress, source, and destination MAC is observed. The Learning Table, Flood Route and Bound (FRB), and Flooding Bound are three key components that form the core of the Bounded Flood SDN Controller. They are explained below.

3.2.1. Learning table

The learning table is a compound structure that maintains decision-making data that is used to build the SDN data plane forwarding database. It performs both Ethernet address learning [22] and root bridge learning. Within an overlay, leaf devices are the initial producers and final consumers of network messages, and they connect to the overlay through their respective root bridge. The table learns of leaf devices and their associated root bridge by observing encountered FRB headers (described in the next section).

Each entry in the root bridge table contains a peer switch descriptor, as observed from the local switch's perspective. The peer switch descriptor comprises the peer node ID, the local port number if the peer is adjacent, and the set of observed leaf devices managed by the peer switch. This data, which is regarded as an incomplete and potentially globally inconsistent descriptor of the peer switch, is treated as soft state and maintained for building on-demand tunnels.

When flow metrics indicate data rates above or below specified thresholds, on-demand tunnels are created or destroyed, respectively. To establish an on-demand tunnel, the host bridge for each leaf device is identified from the root bridge table. The FRB protocol, explained next, guarantees that at least one BF controller in a communicating pair will know of its peer's root bridge.

3.2.2. Flood Route and Bound (FRB)

Flood Route and Bound (FRB) is a custom Ethernet protocol used by BF for SDN peer-to-peer controller communication through overlay tunnels. FRB serves two purposes. First, when the bridge is required to perform a broadcast to another peer bridge, it replaces the operation with an FRB. Hence, the FRB becomes the method for performing duplicate-free broadcasts in an environment with layer 2 link cycles. The FRB is also used to share the local leaf devices hosted by the switch with its adjacent peers. In this role, the FRB includes the Node ID (NID) of the initiating switch (the root NID) and the list of leaf MAC addresses that are connected to the bridge. This exchange is performed whenever a tunnel is established between peers.

In the former role, the FRB header must specify, in addition to the root NID, a bound NID for limiting broadcasts, and the payload that is the original broadcast frame. The bound NID is the identifier of a BF node in the overlay; it indicates to the message recipient how far along the ring the broadcast can be safely propagated and hence prevents message duplication. The bound NID must be recalculated for each of the local bridge ports on which the FRB is transmitted, and this is done using the Flooding Bound Algorithm.

3.2.3. Flooding bound algorithm

This algorithm builds upon FRB messages to handle broadcasts in the presence of cycles in the structured P2P topology, preventing duplicates and allowing all links to be effectively utilized — unlike STP, no links are disabled in BF. The algorithm produces, for each destination NID, the corresponding peer NID that will terminate further propagation of the message, i.e., its propagation boundary. The approach relies on: (1) the previously described structured topology ordered by unique NIDs, (2) the overlay's adjacency list consisting of peer nodes that are connected by an edge, and (3) each adjacent peer bridge that receives the FRB

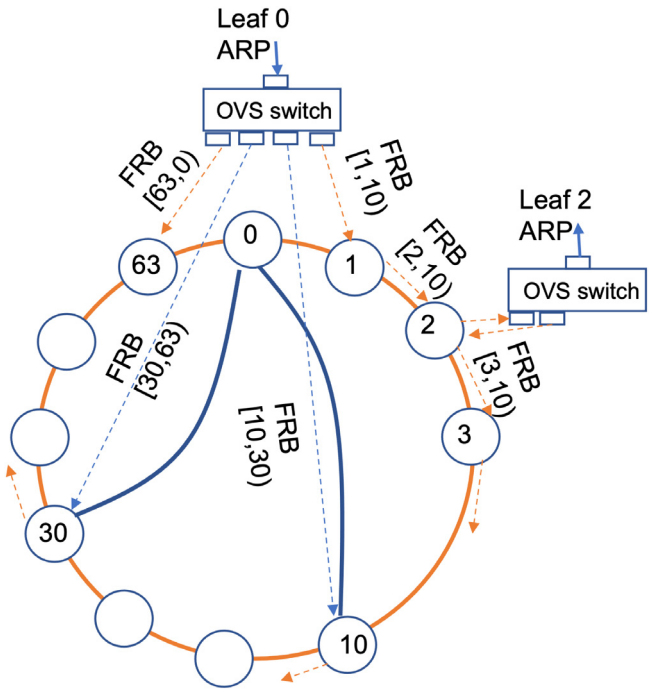

Fig. 2. Example of bounded flood algorithm and FRB messages. For simplicity, this example shows a segment of a small overlay with NIDs from 0–63 (whereas NIDs are 128 bits long). Successor links are colored orange, and long-distance links blue. Dashed lines show FRB messages disseminated in a decentralized fashion as a result from an ARP broadcast sent from leaf device 0. The ARP is encapsulated at the OVS switch in node 0 and FRBs and transmitted on each port with the appropriate bound: e.g. NID = 1 propagates to the [1,10) range (including its leaf devices), while NID = 10 propagates to the [10,30) range. The ARP message is delivered to leaf devices at each node; if a node matches the IP address in the ARP payload and replies, the message traverses the return path back to node NID = 0.

delivers the payload to its leaf devices and forwards the FRB to its peers that lie within bounds.

Each output tuple produced by the algorithm is a closed-open interval $[i, j)$ specifying the recipient i of the message, and the furthest node boundary j to which the recipient may forward the message. This is based on the clockwise distance between two nodes. The NID is an integer representing its distance from position 0, and the furthest possible node is at a distance equal to the maximum value of the node identifier's m-bit address space $D_{max} = 2^m - 1$. Node n_i is considered at distance 0 from itself; its distance to node n_j is given as $d_n = n_j - n_i$ for $n_j > n_i$; otherwise $n_j = n_j + D_{max}$.

To determine the bound, the initiating node selects two adjacent peers: n_1 (the message recipient) and n_2 such that n_2 is the next greater node ID after n_1 in its sorted (by NID) adjacency list. The Flooding Bound tuple is then $[n_1, n_2)$. This procedure is depicted in Fig. 2. This repeats for every such pair in its adjacency list, and for the entry preceding its own, the switch uses its own NID as the bound.

On receipt of an FRB, the SDN controller evaluates its adjacent peers to determine which lie within the received bound and to calculate a new bound for each one. Determining the new bound is done using the same procedure as described above. This distributed algorithm partitions the NID address space into progressively smaller bounds, and results in an acyclic logical forwarding tree that spans the entire graph. Each logical tree is associated with a unique source MAC address, and rooted at the broadcast initiator (NID = 0 in Fig. 2), where multiple such trees can co-exist, cycle-free, in the same overlay. These logical trees are inferred by the flow rules programmed in a decentralized fashion by the SDN controllers at each peer, and are learned in a self-organizing fashion — flows are built/expired dynamically as nodes join and depart the overlay and on demand based on messages including ARP. This is a key differentiating aspect of the scalability of the Bounded Flood protocol compared to STP: BF supports *multiple logical trees that are associated with different MAC addresses and root switches*, while STP only supports a *single* tree.

The flooding bound algorithm ensures that broadcast frames are never duplicated and are delivered to all devices in the overlay, eventually terminating. Furthermore, as FRBs are propagated throughout the overlay, they are tracked at each node's SDN controller to update its local learning table — thereby programming unicast flows across the SDN switches. This information collectively provides a return route across the overlay to the FRB initiator. Taking ARP as an example, the protocol uses FRB controller-to-controller messages to broadcast the ARP request, while also learning and programming the return path. As there are potentially multiple valid paths between any two peers, the network path identified between two nodes is not guaranteed to be the shortest path due to the greedy clockwise routing procedure used for discovery in the flooding process. However, it is bounded by $O((1/k) * log^2(n))$ [33]. The Bounded Flood design imposes no restrictions on the number of links a node can create over its operational lifespan; furthermore, each node can independently and dynamically vary the number of long-distance links it creates. Thus, when $k = logn$, the average overlay switching path is reduced to $O(log(n))$.

3.2.4. Overlay churn

As nodes join and leave the overlay, the restructuring of the topology can potentially disrupt network connectivity within the overlay. This is mitigated by using a configurable number of redundant successor links. In general, to tolerate the concurrent departure of n successors, $n + 1$ successor links are required.

Churn can trim any edge along an active communication path between nodes. As Bounded Flood routing decisions occur independently at each peer, a root bridge will have no indication if the failure involves nodes outside its adjacency list. However, other nodes on the path will detect the failure and will attempt to rediscover a new route to the destination. When no path is known for forwarding the request, the node performs a bounded flood for the forwarding operation. The frame is delivered to the intended recipient at the extra cost of the broadcast.

3.3. Implementation

The Bounded Flood architecture as described in this paper has been successfully implemented and made available as open-source software in EdgeVPN [20]. The implementation is used in the experiments described in subsequent sections. To deploy an instance of the EdgeVPN virtual network, it requires networking data and control plane modules (named Evio-core) that run on each of the edge nodes joining the network, as well as the cloud-based supporting services which facilitates node credentialing and P2P link bootstrapping.

The signaling service allows nodes to authenticate, discover peers belonging to the same network, and use short messages to exchange network endpoints and security keys to bootstrap P2P tunnels. This is leveraged in the form of a service that supports the XMPP [47] (eXtensible Messaging and Presence Protocol) standard, such as the open-source Openfire and eJabberd platforms. NAT traversal services support the negotiation of NAT traversal endpoints for devices that are in private networks and subject to NAT/firewall middleboxes. This is leveraged in the form

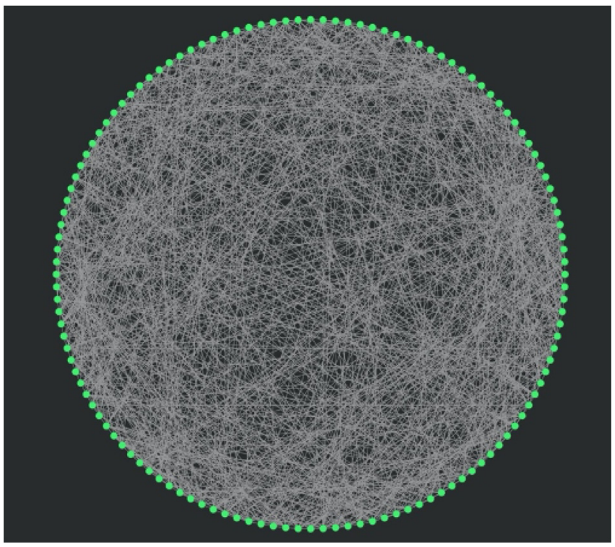

Fig. 3. Visualization of an overlay topology with 128 EdgeVPN peers. Each peer is highlighted as a green dot, while links are shown in white lines.

of services that supports the standards of STUN [48], TURN [49], and ICE [51] protocols, such as the open-source coturn platform. The data plane of each EdgeVPN node performs packet capture and forwarding using software-based switches and IP tunnels. The tunnel abstraction connects two peers and comprises a virtual Network Interface Card (vNIC) on each host and a WebRTC [52] transport. The tunnels at each EdgeVPN node are bound to an Open vSwitch instance which uses OpenFlow to dynamically program its forwarding rules. There are two separate components in the control plane, one responsible for overlay management and the other forwarding rules management. The overlay controller determines peer membership in an overlay and maintains the topology. The SDN controller performs route discovery between peers (by observing ARP broadcasts) and programs the forwarding rules.

Fig. 3 shows a snapshot of the topology of a 128-node overlay deployed using the EdgeVPN software and used in the evaluation of the Bounded Flood technique.

4. Evaluation

4.1. Experiment test cases

Bounded Flood has been implemented in software, as a layer 2 VPN supporting the techniques described in Section 3. The system supports ARP and IP unicast protocols and self-configures an overlay with nodes across multiple edge and cloud endpoints, including private endpoints behind NAT. The following test cases E1–E10 have been conducted to validate functional behavior as per the F1–F6 requirements outlined in Section 1 and assess performance.

E1. The ARP test verifies (F6.a). An ARP is generated using arping, and each BF node records its received ARP requests. The test ensures that every BF node receives at most one ARP request, and the initiator receives the ARP reply.

E2. The reassign IP address test verifies (F6.b). An IP configuration for the same subnet is applied to two nodes and Node 1 is shown to ping Node 2. Node 2 is then powered down and its IP address is reassigned to Node 3. Node 1 is shown to successfully ping the IP address relocated from Node 2 to 3.

E3. Connectivity within partitions test verifies (F1.b, F3.c, F4.b, F5.a). An overlay was instantiated with parameters $n = 64$, $s = 1$, $k = 2$. A node was then selected, and two ping tests were initiated between the node and (1) its successor's successor and (2) the first long-distance linked peer which follows its successor. We shut down the successor node and observe the output of ping.

E4. Resilience against (n-1) failures with n successors test verifies (F1, F4.a). An overlay was instantiated with parameters $n = 64$, $s = 2$, $k = 2$, and the procedure in T4 in repeated. The ping test was then invoked between the node and (1) its peer (pn3) located 3 successor edges away and (2) the first long-distance peer which follows pn3. Both nodes at 1 and 2 successor edges distance were shut down and the output of the ping tests is observed.

E5. Overlay with mixed configuration parameters verifies (F1.a, F6.c). An overlay was configured such that BF nodes had separate amounts of successor and long-distance links, and some had on-demand tunnels enabled or disabled. Nodes 1–8 $s = 1$, $k = \log n$, OND = disabled, nodes 9–16 $s = 2$, $k = 1$, OND = enabled, nodes 17–24 $s = 2$, $k = \log n$, OND = disabled, nodes 24–32 $s = 2$, $k = \log n$, OND = enabled. The functionality was observed.

E6. On-demand tunnel test verifies (F3.c, F3.d). Using the parameters $n = 128$, $s = 2$, OND threshold = 100 MB, total transfer size = 3 GB; 3 overlays were instantiated such that (1) $k = 4$, BF forwarding with on-demand enabled (2) $k = \log(n)$, BF forwarding with on-demand disabled (3) $k = 128$, STP forwarding with MAC learning. We randomly selected 100 pairs of nodes and sequentially ran iperf, followed by ping, between each node pair. We record the bandwidth and average latency for each test in each scenario and noted the differences in average path length between the 2 BF overlays of different k.

E7. The latency test verifies (F2, F7). Overlays were instantiated with parameters $s = 2$, $k = \log n$ and $n = 8, 32, 64, 128$. Each node in the overlay pinged another node in the overlay, generating an FRB in the process. Each node received at least one FRB (broadcast) from every node in the overlay during this process and it allows recording the path length, in switching hops, from every sender. Each node recorded its maximum path length and the average of all path lengths between itself and its neighbors.

E8. Dynamic network (churn) verifies (F2.a, F3.a, F3.b). An overlay is configured with parameters $s = 2$, $k = 2$ and $n = 32$. Half the nodes in the overlay were booted and iperf ran between 2 nodes (without a direct link). We resumed booting of the remaining nodes at 60-s intervals. Next, nodes randomly left and rejoined the overlay until the iperf test completes. We observed the throughput over the duration of the transfer, including any timeouts or failures.

E9. STP vs. BF path utilization test verifies (F3.d). The purpose of this test is to show reduced link contention when using BF. First, an overlay is configured and booted with parameters $n = 225$, $k = \log n$, and STP. STP disables links until a spanning tree is formed. 300 client/server pairs were randomly selected and the bandwidth and latency for each pair were individually measured. The test was repeated on a similar overlay using Bounded Flood instead of STP and with the same node selections.

E10. This test verifies support for unmodified middleware and applications for edge computing use cases. We deploy a Kubernetes cluster across four different sites — including on Raspberry Pi 4 devices on two distinct NAT-ed home networks and with private, non-routable IP addresses. The cluster is used to deploy Docker containers (including Flannel) and allow them to communicate seamlessly, without any changes to Kubernetes, Docker, or Flannel.

E11. This test determines the latency overhead introduced by the overlay. It measures and evaluates (1) the time for creating a new tunnel, (2) the time to relocate an IP address to a new

switch, i.e., when an existing node is replaced by another in the overlay, and (3) the tunneling overhead introduced at each switching node. An overlay is instantiated with parameters $n = 100, s = 1, k = log(n)$. A node n is randomly selected and stopped, reconfigured with a new NID and started. We measure the time for IP connectivity to be restored between n and its successor. Finally, we measure the overhead of tunneling latency introduced at each switching node.

4.2. Testbeds

For experiments **E1–E9**, a testbed was deployed as clusters of Docker containers on physical hosts from CloudLab [53] and Chameleon [54] distributed across the Internet. The choice of Docker captures a target use case for edge computing and validates that the system is operational in current container environments. The processes of each BF node, which included an Open vSwitch (OVS) instance, were executed in a privileged Docker container and created the overlay network within the container's networking namespace. The virtualized network is never directly visible to the host. This approach allowed experiments to scale to networks consisting of hundreds of nodes (up to 200 containers per host). The hosts have enough RAM such that no memory pages are swapped during execution. A single host is used for the experiments, except for **E9**, where 3 physical hosts each with 75 containers are used. The hosts are connected via WAN links which exhibit lower bandwidth and higher latencies compared to the links between containers on the same host. Finally, all the clusters participate in the same virtual Ethernet broadcast domain.

In each container, a secondary bridge was instantiated, and its internal port appropriately configured with an IPv4 address, and a patch link established between it and the OVS. Applications (e.g. iperf) bound a socket to this internal port such that frames were patched over to the local BF bridge, switched across the overlay to the destination BF bridge, patched to the destination application bridge, and delivered to the recipient application.

For **E10**, a heterogeneous ARM-64 cluster with 13 nodes and distributed geographically across four different domains was used. Three of the nodes were EC2 US-West t4 g.medium instances (2 VCPUs. 4 GB memory) ran the Kubernetes control plane and etcd; nine of the nodes were CloudLab m400 (Utah) instances (8 cores and 64 GB memory), and two of the nodes were Raspberry Pi 4 edge devices (4 cores, 4 GB memory) deployed in two different residential ISP providers, behind different NATs. All nodes ran Ubuntu 20.04, Docker 20.10.7, and Kubernetes 1.21.3, and were all connected by EdgeVPN version 21.6.0. The entire Kubernetes cluster was installed via the overlay using Kubespray, Ansible, and SSH.

For **E11**, we have deployed 100 nodes as containers with their respective virtual network interfaces in a single AMD-64 host with 4 hyperthreaded cores and 32 GB RAM. This allows us to capture data that is intrinsic to our system and independent from physical network characteristics.

5. Results and analysis

5.1. Cost of soft state

The SDN data plane forwarding database must store two flow rules per pair of communicating leaf devices. If a bridge has L_{local} leaf devices and there are L_{remote} other leaf devices on the overlay, then the bridge must maintain at most $2 * L_{local} * L_{remote}$ flow entries in the worst case. The flow rules are set to expire over an idle threshold, and typical workloads do not exhibit the worst-case pattern, because each leaf device (e.g. a client or server) only actively communicates with a subset of other leaf devices.

The learning table maintains an ephemeral table for every peer switch and its leaf devices. The data is used to determine flow rules and the endpoints for on-demand tunnels.

The topology state tracks the identifier of each node present in the overlay. Extended structural information is only kept for adjacent peers, which is $2 * (s + k) + o$, where s is the number of successors, k is the number of long-distance and o is the number of on-demand links.

5.2. Join/departure cost

When a node attempts to join an overlay, an XMPP sign-in presence message is exchanged with each online peer for an $O(2n)$ cost. Creating a single link is a constant $O(1)$ cost that includes the messages related to Interactive Connectivity Establishment (ICE) [51] and the exchange of endpoint data between the two hosts for NAT traversal. Each node creates s successors and k long-distance links and receives at most the same amount of incoming links, resulting in $2*(k+s)$ links for the join operation. The join cost is $O(2n + 2(k + s))$.

When a node leaves the overlay, s successor links must be repaired. If the departure resulted in a change of $\lfloor log(n) \rfloor$, n links in the overlay are discarded and relinking occurs. However, only one long-distance link per node possibly requires relinking. The cost to leave is $O(n + s)$ in the worst case, and otherwise $O(s)$ on average.

5.3. Switching overhead

To measure switching overhead in experiment **E11**, we have instrumented virtual NIC devices to gather timestamps, and used end-to-end ping round-trip time (RTT) measurements. For this network, the average link creation time was 2.4 s and the average time to resolve IP relocation was 6.15 s.

The switching overhead is primarily due to the user-space processing in the tunneling module — responsible for encapsulation, decapsulation, encryption, and decryption. In the **E11** testbed, round trip times are dominated by the user-space processing; we also measured the kernel-level processing due to Open vSwitch, and it is negligible compared to the user-space tunneling module. We performed experiments to measure the latency between a pair of nodes that were separated by 0 to 4 switching nodes. Increasing the number of switching nodes on the path increases the RTT, and the increase at each step is attributed to the overhead of the additional switching node. It measures the combined switching time for the request and response at that node. Hence, the average switching time of a node is one half the increase in RTT.

The measured switching overhead was 0.37 ms per switched packet. While this overhead is negligible over WAN connections, its impact is more noticeable on low-latency LAN links.

5.4. Route discovery cost

BF is used as a replacement to Ethernet broadcast for route discovery between a pair of communicating hosts. A single message must be delivered from the initiator to each participant in the overlay, but only once. Hence, the message cost of route discovery is the same as broadcast and is simply $O(n)$. Multiple successor links provide redundancy for fault tolerance. The long-distance links are used for efficiency and are not necessary for correctness.

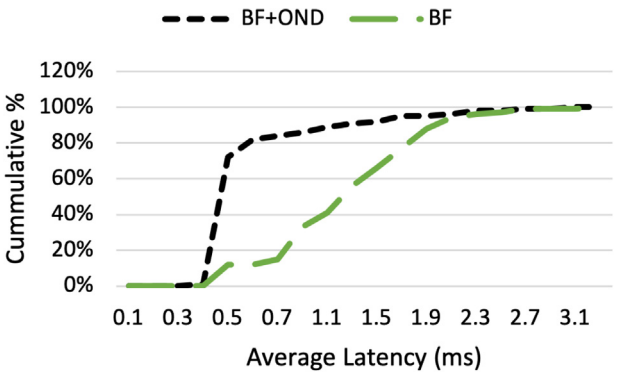

Fig. 4. Cumulative distribution of average latency: BF and BF+OND (on-demand tunnels). Larger values are better as they indicate a larger percentage of paths with lower latency. Latency is measured after the OND tunnel has been established. BF+OND improves the distribution of latency compared to BF; for example, with BF+OND, around 80% of measured samples have a latency *smaller* than 0.5 ms, while for BF without OND, 80% of samples have a latency *larger* than 0.5 ms.

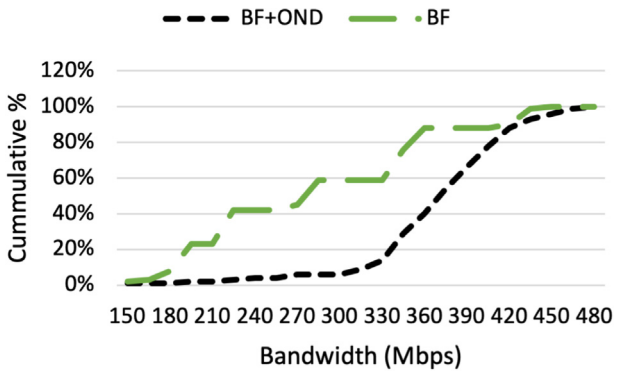

Fig. 5. Cumulative distribution of bandwidth: BF and BF+OND (on-demand tunnels). Smaller values are better as they indicate a smaller percentage of paths with low bandwidth. BF+OND improves the distribution of bandwidth compared to BF; for example, with BF+OND, around 80% of measured samples have bandwidth *larger* than 330 Mbps, while for BF without OND, around 60% of samples have bandwidth *smaller* than 330 Mbps.

5.5. Verification of correctness tests

All tests for correctness, **E1–E5** completed as expected. **E1:** Inspection of the Ethernet broadcast of ARP using BF indicated a single request/response exchange and the host was found via the broadcast. **E2:** The IP address was successfully located via ping before and after its relocation to a different node and MAC address in the overlay. **E3:** Ping to the second successor failed when the 1st successor was shut down; however, ping to the first long-distance node was uninterrupted. This indicates a partitioning in the network between the failed node and up to the 1st long-distance peer. As the network repaired its topology the failed transmission resumed. **E4:** Using s = 2 prevents the network partitioning experienced in the previous test, but only against a single node being shut down. When two adjacent nodes are shut down, the transmission interruption manifests again. As a note, the overlay tolerated two nodes shutting down once they were not immediate successors. **E5:** The overlay converged to a stable state where, in the absence of churn, no new links were being created. ARP, ICMP, and TCP between nodes were tested successfully.

5.6. On-demand tunnels

Experiment **E6** validates the benefits of using on-demand tunnels with BF (Figs. 4 and 5) and shows that on-demand (OND) tunnels have a detrimental effect on STP (Fig. 6).

Bounded Flood with on-demand links (BF+OND) shows improvement in both bandwidth and latency, despite fewer links available in the overlay and a longer average path length than BF. For instance, inspecting an inflection point in Fig. 4 at 0.6 ms, we observe in overlay BF+OND, 82% of latencies below 0.6 ms, while only 12% were below 0.6 ms in overlay BF; a 6.8x improvement. Additionally, Fig. 5 shows 96% of bandwidth tests were over 330 Mbps in overlay BF+OND as opposed to 41% in overlay BF, a 2.4x improvement.

Overall, STP suffers from disruptions when the topology changes (Fig. 6), while BF can gracefully adapt. Overlay BF+OND (OND enabled) is configured to use fewer long-distance links than Overlay BF (OND disabled) and shows the expected increase in overlay average path length. Overlay BF+OND average path length is 4.32 with a mean deviation of 0.31, while overlay BF average path length is 3.41 with a mean deviation of 0.27. It takes

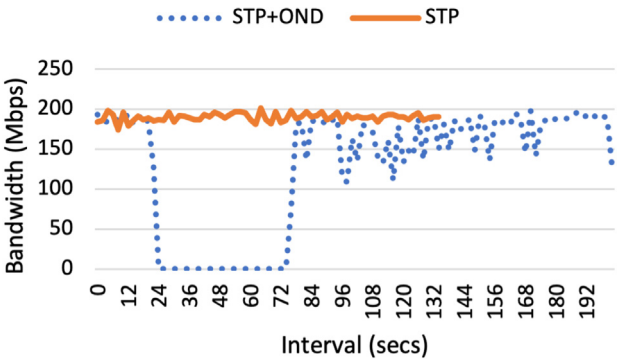

Fig. 6. Bandwidth variation: STP and STP+OND (on-demand tunnels). When a new link is added, the STP spanning tree must be recalculated, and network transmission is interrupted until the process completes. As this interruption takes several seconds (around$t$ = 24 to t = 72 s), on-demand tunnels are detrimental to performance in STP.

approximately an additional switching hop on average to deliver a frame between each pair of nodes.

By employing on-demand (OND) tunnels, a BF+OND overlay can achieve better bandwidth and latencies than without it, while using fewer long-distance links. As tunnels incur an ongoing maintenance cost, using a lower k reduces the associated overhead for each node.

5.7. Network switching latency (path length in hops)

The average switching cost of the overlay is $O((1/k) * log^2(n))$ which simplifies to $O(log(n))$ when k is configured as $log(n)$. From experiment **E7**, each node records the maximum number of hops between itself and every other peer, and the average of all path lengths to every peer.

The results in Table 1 indicate that while the maximum hop count observed at each node can exceed the expected bound, for an overlay with 128 nodes, only 3 paths had a length of 12, and approximately three-quarters of the paths had a length less than 9.

Fig. 7 shows the network average path length scales sub-linearly with the number of nodes in the overlay; for the network sizes evaluated, it is approximately $1/2log(n)$ bound. As switching hops varies with the path length, it can be bound to a function of overlay size. Furthermore, as BF handles Ethernet loops, a direct edge can be placed between nodes to reduce the cost of switching

Table 1
Maximum path lengths. Frequency and cumulative percentages, in an overlay of size n = 128, of each node's maximum path length, i.e., its furthest neighbor.

Max hops	Frequency	Cummulative %
5	1	0.78%
6	14	11.7%
7	32	36.7%
8	21	53.1%
9	27	74.2%
10	22	91.4%
11	8	97.7%
12	3	100.0%

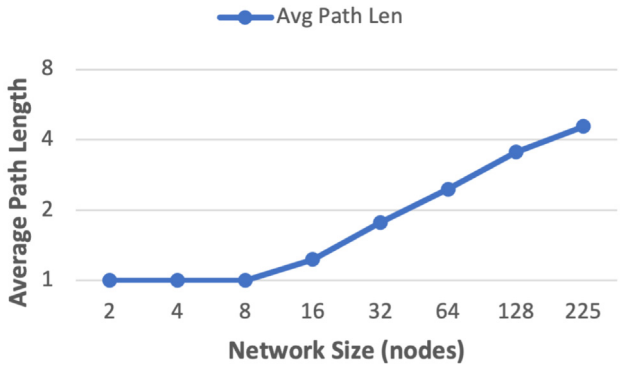

Fig. 7. Average path length (log scale) vs. network size. In this experiment, each node reports its average path length as the sum of all discovered path lengths divided by the number of paths. The network-wide average path length is the average across all nodes.

to a single hop. This supports the claim that BF works well for both small and large overlays.

5.8. Dynamic network churn

Experiment **E8** validates that BF forwarding remains functional in a dynamic and changing topology. Within the specifications of the test, the switching flows can be reprogrammed, and the topology updated without impact to existing TCP streams. From a converged overlay, 8 nodes were randomly selected and sequentially started and stopped, with a 15-s interval between each event, while active iperf flows were present among other nodes. While there were fluctuations in throughput, no iperf transmission failure or timeout was observed

5.9. BF vs. STP performance

Experiment **E9** validates that BF allows multiple paths to be safely utilized to deliver a performance benefit over STP. From Fig. 8, BF shows a greater occurrence of low latency paths compared to STP, with 97% more nodes with a latency less than 36 ms. The maximum bandwidths reported in two similarly configured overlays were similar, but Fig. 9 shows that BF had more paths with higher bandwidth than STP. Examining inflection point at 60 Mbps, approximately 37% of BF measured bandwidth is greater, while only 18% of STP bandwidth was above 60 Mbps – a 2.2x increase in the number of such paths. Compared to STP, BF exhibits better latencies and bandwidth, experiencing reduced link contention and fewer switching hops, as a result of utilizing all available links for transmission. STP must selectively disable links to create its spanning tree, which has the effect of creating longer paths with increased sharing

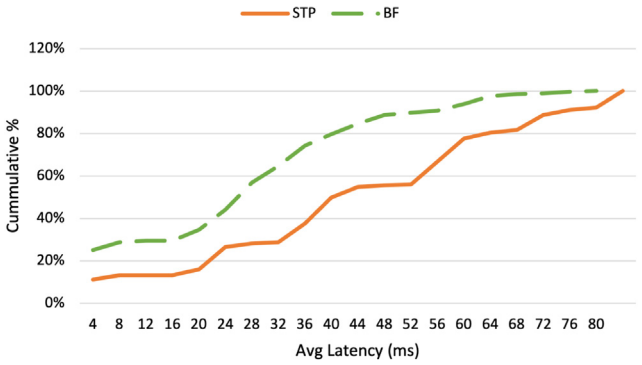

Fig. 8. Cumulative distribution of latency for BF (green) vs. STP (orange). Larger values are better as they indicate a larger percentage of paths with lower latency.

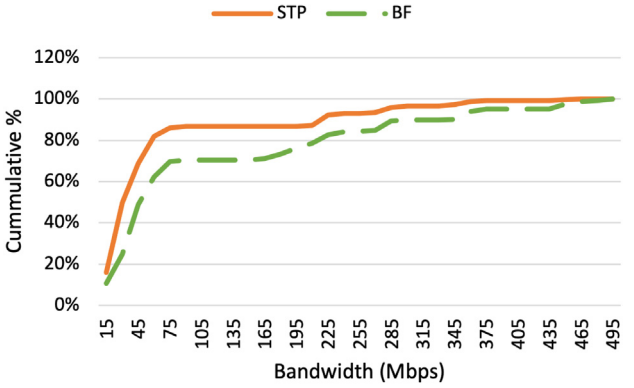

Fig. 9. Cumulative distribution of latency for BF (green) vs. STP (orange). Smaller values are better as they indicate a smaller percentage of paths with low bandwidth.

5.10. Kubernetes edge cluster

Experiment **E10** demonstrates that BFenables unmodified middleware and applications, and quantifies overheads in network latency and throughput. Kubernetes, Docker, and Flannel were deployed successfully across the distributed cluster using Kubespray — only a single YAML configuration line was customized to set Flannel to bind to the virtual network bridge. The functional behavior of the cluster was verified by executing kubectl commands to deploy daemon sets for Flannel and iperf3 pods.

While the cluster had 13 deployed nodes, we focus on three of them for the quantitative evaluation: nodeAWS (control-plane AWS EC2 instance), nodePiA, and nodePiB (Pi 4 edge devices on residential ISP's A and B). The nodePiA, nodePiB devices have private local IP addresses that are translated dynamically by their respective NATs, whereas nodeAWS has a private local IP address that is statically mapped to a public address during instance startup.

The first quantitative experiment considers the round-trip latency (100 ICMP ping messages with 1 s wait times) between nodePiA and nodeAWS for three different scenarios: physical (nodePiA pings nodeAWS using the AWS-exposed address), BF (pings are sent over the BF interface), and Flannel (pings are sent over Flannel, which encapsulates over BF). The results, summarized in Table 2, show a negligible difference in average ping

Table 2
ICMP round-trip latency (ms) between nodePiA and nodeAWS. Avg/Stdev across 100 samples.

Interface	Min	Max	Avg	Stdev
Physical	88	106	92.8	2.20
BF	89	100	91.5	1.73
Flannel+BF	89	105	92.4	2.68

Table 3
iperf3 throughput (Mbit/s) between nodePiA and nodeAWS. Median/Avg/Stdev across 300 samples.

Interface	Median	Avg	Stdev
Physical	31.5	31.1	9.05
BF	31.5	29.8	8.04
Flannel+BF	31.5	29.1	7.25

Table 4
iperf3 throughput (Mbit/s) between nodePiA and nodePiB. Median/Avg/Stdev across 300 samples.

Interface	Median	Avg	Stdev
Physical	N/A	N/A	N/A
BF	33.9	32.8	6.12
Flannel+BF	32.3	32.4	5.36

latency that is well within the standard deviation of the physical network path between the devices

The second experiment considers the throughput (measured using TCP/IP transfers using the iperf3 benchmark) between nodePiA and nodeAWS. Three 100 s runs were performed for each scenario, and throughput was sampled/recorded by iperf3 every second. The results, summarized in Table 3, also show a small difference in average throughput that is well within the standard deviation of the physical network path between the devices

Next, we evaluate iperf3 throughput (same method as above) between nodePiA and nodePiB. Table 4 shows that BF delivers a 30 Mbps+ virtual network flow between the two nodes, where direct connectivity over the physical network is not applicable (N/A) because the private IPs are not routable on the public Internet. With BF, a NAT-traversing STUN tunnel is seamlessly created between nodePiA and nodePiB to carry virtual network traffic.

6. Conclusions

While contemporary IoT applications use a tiered architecture — from device to edge to cloud, this does not imply that the overlay topology must necessarily be a tree. The foundational structured overlay topology with the ring and long-distance nodes used in BF offers several key benefits: self-organization in dynamic environments (nodes join/leave independently), scalability (O(log(N) routing)), and availability (no central points of failure), and additional on-demand links can respond dynamically to capture the traffic patterns of a tiered architecture, in essence allowing for traffic-driven tree topologies to self-organize atop the foundational topology.

The flexibility of the system allows peers to be parameterized independently and reflects the conditions encountered in distributed and heterogeneous operational environments of the fog. Nodes within the overlay may very likely be owned and controlled by independent entities, each with its distinct configuration criteria. Nonetheless, we have demonstrated the ability to deploy unmodified software under such conditions.

There are many challenges to building and maintaining a resilient overlay, and it must handle the traditional P2P vulnerabilities to rogue members. Even with good faith participants, transient environmental problems can delay timely repairs to the overlay. Design choices that are practical to implement are critical to a system such as this. The importance of resilience within the overlay and the disruptive impact of churn was a major factor in selecting Symphony over Chord. Chords must be maintained at each log n node, whereas Symphony requires only that the less stringent conditions of its harmonic function are met, which results in fewer changes to existing links.

Kademlia [34], which offers comparable performance characteristics to Chord and Symphony, can also be reasonably implemented in an EdgeVPN/BoundedFlood system. The similarities with address space, identifiers, and routing for node lookup allow critical abstractions to map from one to the other. For example, the KAD routing table translates to the EdgeVPN overlay controller's adjacency list, and Kademlia's concurrent, recursive node lookup operation translates to the BF broadcast operation.

Our experiments show that using values of $k = 4$ vs. $k = 7$ when $n = 128$ results in an additional switching cost of approximately one extra hop. This setup can be used in environments where the increased switching latency can be tolerated for the benefit of reduced tunnel overhead.

There are clear benefits that have been illustrated when using BF versus STP. STP must disable edges in the presence of loops to create the spanning tree; hence, it cannot utilize all available links in the topology. Using fewer links result in longer paths with increased latency, and the utilized links experience higher contention as more traffic is routed over them. Additionally, the addition or removal of a link in the overlay requires interrupting network transmission until a new spanning tree is evaluated. As such, no benefits are derived from on-demand edges with STP. In topologies with multiple layer 2 loops, BF outperforms STP providing lower latency and increased available throughput.

CRediT authorship contribution statement

Kensworth Subratie: Conceptualization, Methodology, Software, Investigation, Experiments, Original draft preparation. **Saumitra Aditya:** Conceptualization, Software. **Renato J. Figueiredo:** Supervision, Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All the source code is available as open-source code on GitHub. Data from experiments will be provided on request.

Acknowledgments

This material is based upon work supported by the National Science Foundation, USA under Grants OAC-2004441, OAC-2004323, and CNS-1951816. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] N. Gershenfeld, R. Krikorian, D. Cohen, The internet of things, Sci Am. 291 (4) (2004) 76–81, http://dx.doi.org/10.1038/scientificamerican1004-76.

[2] E.A. Lee, Cyber physical systems: Design challenges, in: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), 2008, pp. 363–369, http://dx.doi.org/10.1109/ISORC.2008.25.

[3] K. Ashton, That 'internet of things' thing, RFID J. 22 (7) (2009) 97–114.

[4] M. Satyanarayanan, Pervasive computing: vision and challenges, IEEE Pers. Commun. 8 (4) (2001) 10–17, http://dx.doi.org/10.1109/98.943998.

[5] B. Zhang, N. Mor, J. Kolb, D.S. Chan, N. Goyal, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, J. Kubiatowicz, The cloud is not enough: Saving iot from the cloud, in: Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing, HotCloud '15, 2015, p. 21.

[6] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. 98 (2019) 289–330, http://dx.doi.org/10.1016/j.sysarc.2019.02.009.

[7] Internet society - state of IPv6 deployment 2018, 2022, https://www.internetsociety.org/resources/2018/state-of-ipv6-deployment-2018, Accessed May 4, 2022.

[8] Google IPv6 statistics, 2022, https://www.google.com/intl/en/ipv6/statistics.html, Accessed May 4, 2022.

[9] M.A. Tamiru, G. Pierre, J. Tordsson, E. Elmroth, Mck8s: An orchestration platform for geo-distributed multi-cluster environments, in: ICCCN 2021 - 30th International Conference on Computer Communications and Networks, Athens, Greece, 2021, pp. 1–12, URL https://hal.inria.fr/hal-03205743.

[10] F. Palmieri, VPN scalability over high performance backbones evaluating MPLS VPN against traditional approaches, in: Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003, 2003, pp. 975–981 vol.2, http://dx.doi.org/10.1109/ISCC.2003.1214243.

[11] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646, http://dx.doi.org/10.1109/JIOT.2016.2579198.

[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: Enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74, http://dx.doi.org/10.1145/1355734.1355746.

[13] N. Feamster, J. Rexford, E. Zegura, The road to SDN: An intellectual history of programmable networks, SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 87–98, http://dx.doi.org/10.1145/2602204.2602219.

[14] Enterprise container platform, 2022, https://www.docker.com, Accessed May 4, 2022.

[15] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade, ACM Queue 14 (1) (2016) 70–93, http://dx.doi.org/10.1145/2898442.2898444.

[16] Flannel, 2022, https://github.com/flannel-io/flannel, Accessed May 4, 2022.

[17] Project calico, 2022, https://www.tigera.io/project-calico/, Accessed May 4, 2022.

[18] Weave net, 2022, https://www.weave.works/docs/net/latest/overview/, Accessed May 4, 2022.

[19] K. Subratie, S. Aditya, V. Daneshmand, K. Ichikawa, R. Figueiredo, On the design and implementation of IP-over-P2P overlay virtual private networks, IEICE Trans. Commun. E103-B (1) (2020).

[20] EdgeVPN.io: Open-source VPN for edge computing, 2022, https://edgevpn.io, Accessed May 4, 2022.

[21] C.E. Spurgeon, Ethernet: The Definitive Guide, O'Reilly Media, Inc, 2000.

[22] C.E. Spurgeon, J. Zimmerman, Ethernet Switches: An Introduction to Network Design with Switches, O'Reilly Media, Inc, 2013.

[23] IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges, IEEE Std 8021D-2004 Revis. IEEE Std 8021D-1998, 2004.

[24] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, Portland: A scalable fault-tolerant layer 2 data center network fabric, in: Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication, SIGCOMM '09, Association for Computing Machinery, New York, NY, USA, 2009, pp. 39–50, http://dx.doi.org/10.1145/1592568.1592575.

[25] Ryu SDN framework, 2022, https://ryu-sdn.org/, Accessed May 4, 2022.

[26] Z. Yang, Y. Cui, B. Li, Y. Liu, Y. Xu, Software-Defined Wide Area network (SD-WAN): Architecture, advances and opportunities, in: 2019 28th International Conference on Computer Communication and Networks (ICCCN), 2019, pp. 1–9, http://dx.doi.org/10.1109/ICCCN.2019.8847124.

[27] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: Challenges and opportunities for innovations, IEEE Commun. Mag. 53 (2) (2015) 90–97, http://dx.doi.org/10.1109/MCOM.2015.7045396.

[28] D. Andersen, H. Balakrishnan, F. Kaashoek, R. Morris, Resilient overlay networks, SIGOPS Oper. Syst. Rev. 35 (5) (2001) 131–145, http://dx.doi.org/10.1145/502059.502048.

[29] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Trans. Netw. 11 (1) (2003) 17–32, http://dx.doi.org/10.1109/TNET.2002.808407.

[30] H. Eriksson, MBONE: The multicast backbone, Commun. ACM 37 (8) (1994) 54–60, http://dx.doi.org/10.1145/179606.179627.

[31] B.Y. Zhao, L. an Huang, J. Stribling, S.C. Rhea, A.D.J. Joseph, J.D. Kubiatowicz, Tapestry: A resilient global-scale overlay for service deployment, IEEE J. Sel. Areas Commun. 22 (1) (2004) 41–53.

[32] A. Rowstron, P. Druschel, Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems, in: R. Guerraoui (Ed.), Middleware 2001, Springer Berlin Heidelberg, 2001, pp. 329–350.

[33] G.S. Manku, M. Bawa, P. Raghavan, Symphony: Distributed hashing in a small world, in: Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems, 2003.

[34] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer information system based on the XOR metric, in: Peer-To-Peer Systems, Springer Berlin Heidelberg, 2002, pp. 53–65.

[35] X. Jiang, D. Xu, VIOLIN: Virtual internetworking on overlay infrastructure, in: J. Cao, L.T. Yang, M. Guo, F. Lau (Eds.), Parallel and Distributed Processing and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 937–946.

[36] A.I. Sundararaj, A. Gupta, P.A. Dinda, Dynamic Topology Adaptation of Virtual Networks of Virtual Machines, LCR '04, Association for Computing Machinery, New York, NY, USA, 2004, pp. 1–8, http://dx.doi.org/10.1145/1066650.1066665.

[37] D. Joseph, J. Kannan, A. Kubota, I. Stoica, K. Wehrle, OCALA: An architecture for supporting legacy applications over overlays, in: 3rd Symposium on Networked Systems Design & Implementation (NSDI 06), USENIX Association, San Jose, CA, 2006.

[38] J. Maassen, H.E. Bal, Smartsockets: Solving the connectivity problems in grid computing, in: Proceedings of the 16th International Symposium on High Performance Distributed Computing, HPDC '07, Association for Computing Machinery, New York, NY, USA, 2007, pp. 1–10, http://dx.doi.org/10.1145/1272366.1272368.

[39] Romana networking, 2022, https://romana.readthedocs.io/en/latest/Content/networking.html, Accessed May 4, 2022.

[40] Network and security virtualization software platform - NSX, 2022, https://www.vmware.com/products/nsx.html, Accessed May 4, 2022.

[41] MidoNet - virtualized networking for public and private clouds, 2022, https://github.com/midonet/midonet, Accessed May 4, 2022.

[42] M. Mahalingam, et al., Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks, 2014, https://www.rfc-editor.org/info/rfc7348, Accessed May 4, 2022.

[43] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, V. Jain, An intent-based approach for network virtualization, in: Proceedings of the IFIP/IEEE Int. Symp. on Integrated Network Management, 2013, pp. 42–50.

[44] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E.C. Zermeno, E. Rubow, J.A. Docauer, J. Alpert, J. Ai, J. Olson, K. DeCabooter, M. de Kruijf, N. Hua, N. Lewis, N. Kasinadhuni, R. Crepaldi, S. Krishnan, S. Venkata, Y. Richter, U. Naik, A. Vahdat, Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization, in: 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), USENIX Association, Renton, WA, 2018, pp. 373–387.

[45] Zerotier protocol design whitepaper, 2022, https://docs.zerotier.com/zerotier/manual/, Accessed Aug 26, 2022.

[46] SoftEther VPN project, 2022, https://www.softether.org/, Accessed Aug 26, 2022.

[47] The universal messaging standard, 2022, https://xmpp.org, Accessed May 4, 2022.

[48] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, STUN - simple traversal of user datagram protocol (UDP) through network address translators (NATs), in: Internet Engineering Task Force (IETF), RFC 3489, 2003.

[49] R. Mahy, P. Matthews, R. J., Traversal using relays around NAT (TURN): Relay extensions to session traversal utilities for NAT (STUN), in: Internet Engineering Task Force (IETF), RFC 5766, 2010.

[50] J. Kleinberg, The small-world phenomenon: An algorithmic perspective, in: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, STOC '00, Association for Computing Machinery, New York, NY, USA, 2000, pp. 163–170, http://dx.doi.org/10.1145/335305.335325.
[51] J. Rosenberg, Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols, 2010, https://www.rfc-editor.org/rfc/rfc5245.txt, Accessed May 4, 2022.
[52] Real-time communication for the web, 2022, https://webrtc.org, Accessed May 4, 2022.
[53] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, P. Mishra, The Design and Operation of Cloudlab, in: USENIX ATC '19, USENIX Association, USA, 2019, pp. 1–14.
[54] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, P. Ruth, Chameleon: A scalable production testbed for computer science research, in: Contemporary High Performance Computing, 2019, pp. 123–148, http://dx.doi.org/10.1201/9781351036863-5.

Dr. Kensworth Subratie is a Post-Doctoral Researcher in the College of Engineering (Electrical and Computer Engineering) at the University of Florida. He graduated with a Ph.D. in ECE From the University of Florida in 2019. His research interests lie in fog computing, autonomic and peer-to-peer systems, virtual networks and software system design.

Dr. Saumitra Aditya is a Senior Software Engineer at Akamai Technologies. He graduated with a Ph.D. in ECE From the University of Florida in 2020. His research interests broadly span networked systems and their applications in realization of smart communities.

Dr. Renato J. Figueiredo is a Professor of Electrical and Computer Engineering at the University of Florida, where he leads research on topics that include resource virtualization (virtual machines, networks, and storage), cloud computing, peer-to-peer systems, autonomic computing, high performance and high-throughput computing, and network overlay applications in distributed systems. He graduated with a Ph.D. in ECE from Purdue University in 2001, joined Northwestern University as an Assistant Professor in 2001, and subsequently the University of Florida as an Assistant Professor in 2002. His research has been funded by government and industry sponsors that include the National Science Foundation, Intel Corp., IBM Corp., NOAA, and NASA. He has published over 140 technical papers in conferences and journals, and served as Technical Program Committee Co-Chair for the International Conference on Autonomic Computing (ICAC, 2010) and the International Symposium on High-Performance Parallel and Distributed Computing (HPDC, 2013). He served as site co-director of the NSF Industry/University Cooperative Research Center (I/UCRC) for Cloud and Autonomic Computing (CAC).