Aalto University

OPEN ACCESS

This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Varis, Nuutti; Manner, Jukka; Särelä, Mikko; Kiravuo, Timo

DBridges: Flexible Floodless Frame Forwarding

Published in:
Computer Networks

DOI:
10.1016/j.comnet.2013.08.007

Published: 01/01/2013

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY-NC-SA

Please cite the original version:
Varis, N., Manner, J., Särelä, M., & Kiravuo, T. (2013). DBridges: Flexible Floodless Frame Forwarding. *Computer Networks*, *57*(17), 3601-3616. https://doi.org/10.1016/j.comnet.2013.08.007

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Computer Networks 57 (2013) 3601–3616

Contents lists available at ScienceDirect

Computer Networks

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

DBridges: Flexible floodless frame forwarding

Nuutti Varis *, Jukka Manner, Mikko Särelä, Timo Kiravuo

Aalto University School of Electrical Engineering, Department of Communications and Networking, P.O. Box 13000, 00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 3 February 2013
Received in revised form 31 July 2013
Accepted 8 August 2013
Available online 23 August 2013

Keywords:
Ethernet
Dht
Network architecture

ABSTRACT

Ethernet has become the de facto layer 2 transmission technology, partly due to ease of use and cost efficiency. The cores of most data centers around the world are based on Ethernet, and large access and core networks are built without IP routing. The inherent simplicity of Ethernet has several drawbacks, including the overhead of network-wide broadcasting and the use of the spanning tree protocol. The academia and the industry have presented numerous solutions to overcome the limitations of Ethernet. Many of these solutions have deployment constraints and often force a change of the whole network.

In this paper, we present DBridges, an efficient solution for building large-scale Ethernets based on the IETF-driven RBridges and SEATTLE. DBridges provide loop-safety, dramatic decrease in broadcast traffic, and resiliency and graceful failover without affecting the plug-and-play vision of Ethernet. The key contributions of our solution are that we are backward compatible with RBridges and can support incremental deployment. We analyze and show these benefits in detail, and present various performance measurements using our proof-of-concept implementation. We estimate that with DBridges it is possible to increase the size of Ethernet broadcast domains at least tenfold.

© 2013 The Authors. Published by Elsevier B.V. Open access under CC BY-NC-SA license.

1. Introduction

Ethernet was originally envisioned to be a simple, cost-effective, and zero configuration transmission technology. Over the years, Ethernet has become the prevalent link layer network technology in the world, with deployments ranging in scope from residential homes to massive data centers. To support the deployment diversity, Ethernet has been extended with loop prevention mechanisms, Virtual Local Area Networks (VLANs), and various other technologies. As a result, today's cost-effective Ethernet bears little resemblance to the original vision of a zero-configuration network technology.

Ethernet networks have several well-known scaling issues that limit the size and topology of the network. Many scalability issues are related to the operation of the Spanning Tree Protocol (STP) [1]. Its primary responsibility is to build a loop-free forwarding tree from the network topology, that switches use to forward Ethernet frames between hosts in the network. A large issue in current, high-volume Ethernet networks is that the protocol builds a single forwarding tree to use in the Ethernet network, that forwards traffic using a single path out of all the available paths in more mesh-like topologies. Additionally, the original protocol design itself suffers from instability in large Ethernet topologies. Finally, while STP offers a loop free forwarding topology in normal operation, there are cases, such as hardware failure, where the loop free operation is not guaranteed. This may lead to uncontrolled frame propagation in the network, causing end to end service for hosts to be disrupted throughout the Ethernet network.

* Corresponding author. Tel.: +358 947025480.
E-mail addresses: nuutti.varis@aalto.fi (N. Varis), jukka.manner@aalto.fi (J. Manner), mikko.sarela@aalto.fi (M. Särelä), timo.kiravuo@aalto.fi (T. Kiravuo).

1389-1286 © 2013 The Authors. Published by Elsevier B.V. Open access under CC BY-NC-SA license.
http://dx.doi.org/10.1016/j.comnet.2013.08.007

Arguably, the most significant scalability issue in Ethernet networks is also one of the main motivators for the ubiquitous use of Ethernet in modern networks. Ethernet uses flooding to reach the recipient in the network, which guarantees end to end service without any host configuration, and simplifies switch design which leads to a cost effective solution. Flooding is minimized in switches by learning the locations of the hosts in the network from received frames. This allows the switch to forward the frame on a specific link, instead of flooding it on a number of links. Unfortunately, in large or highly dynamic networks, the number of unknown hosts can be significant. Additionally, the size of the Ethernet network is limited by the size of the learning database in devices, that scales with the number of active hosts in the network.

Flooding is also amplified by IP. Hosts must resolve the higher layer IP addresses to Ethernet addresses using the Address Resolution Protocol (ARP) [2] or the Neighborhood Discovery (ND) [3] protocol. Both protocols function by flooding the address requests throughout the network, significantly increasing the flooded traffic in large deployments. In addition, the network protocol addressing is typically automatically configured on hosts via a separate protocol such as the Dynamic Host Configuration Protocol (DHCP) [4] that also makes use of flooding.

It has also been recently argued that the original vision of the Ethernet might not even be feasible in large-scale networks with arbitrary topologies. More specifically, PortLand [5] argues that a solution that satisfies loop safety, broadcast scalability, resiliency, and plug-and-play characteristics may not be possible without restricting the topology of the Ethernet or requiring changes to end hosts.

The IETF TRILL working group has introduced its own solution called Routing Bridges (RBridge) [6,7], which mainly targets loop safety, the removal of the scalability and efficiency issues of STP, and enabling the core of the network to scale to larger deployments. Unfortunately, RBridges do not solve the broadcast-driven scalability issues in large Ethernets.

At the same time, the SEATTLE [8] design proposes a radical departure from conventional Ethernet, solving the broadcast-driven scalability issue of large Ethernets with a DHT-based approach (discussed in Section 4). SEATTLE is designed for enterprise networks, where the operators typically have a higher degree of control over the hosts that connect to it. To use the design in generic Ethernet networks, some of its operational restrictions must be relaxed.

Overall, RBridges or SEATTLE alone do not offer a complete solution to all of the issues plaguing large Ethernet networks. We attempt to comprehensively solve the scalability issues of Ethernet by extending routing bridges with the DHT concept from SEATTLE. We call the DHT-enhanced RBridge "*DBridge*" to differentiate it from unmodified standard-based RBridges. The use of the DHT as presented in SEATTLE effectively removes broadcast traffic from the network.

We chose to spend much of our effort in designing a system that can be easily deployed in existing Ethernets. DBridges work with the existing install base of hosts and Ethernets switches, including RBridges, can be deployed incrementally, and supports graceful fail-over in the case of resource starvation or network faults.

Thus, the contributions of our work are the *design, implementation, and analysis of a new RBridge-based solution to scalable Ethernet networking*. The primary target environment examined in our work is a data center, however we also show that our solution offers the same benefits in a more generic Ethernet-based network. Using our proof-of-concept implementation we thoroughly evaluate the DHT scheme as a part of the RBridges, and verify through measurements that we retain the key benefits of the approach:

- Up to 99% of the broadcasting in the network is eliminated,
- DHT specific processing is distributed flexibly to several different devices in the network, and
- the agressive cleanup of MAC learning tables in the network allows significantly larger Ethernet deployments without edge switch MAC table overload,
- various network faults and resource starvation are handled gracefully without loss of host connectivity, and
- deployment can be done incrementally with each new node removing a part of the broadcast traffic.

The rest of the paper is structured as follows. First, Section 2 describes some of the previous works that have attempted to solve the Ethernet scalability issues. Next, in Sections 3 and 4, we go over the IETF routing bridges standard, and the SEATTLE design in more detail. Next, Section 5 describes the introduction of the SEATTLE one-hop DHT scheme in routing bridges, and our architectural changes to this scheme. Section 6 describes our software prototype for RBridges and DBridges and Section 7 describes our testing environment and tools in detail. In Section 8, we go over the results of our testing and analyze some of our claims in detail, using two different traffic models. Finally, Section 9 concludes the paper.

2. Related work

Scaling Ethernet broadcast domains beyond a few hundred hosts has been proposed in the past [9] with projects such as the Smartbridge [10], while routing based on a flat namespace has been previously explored in work such as ROFL [11]. There has been a significant research effort to find scalable solutions for data center networking that would solve efficiency, safety, and size issues of the Ethernet network.

The SEATTLE [8] design has been the main inspiration for our work. DBridges are a fusion of the SEATTLE one-hop DHT design and TRILL. SEATTLE has several shortcomings, such as loop-safety and switch forwarding table state, both of which are either completely or partially mitigated by TRILL and our DBridge design. Leveraging TRILL allows DBridges to function alongside RBridges and STP-based switches, while solving one of the major issues in current Ethernet networks.

Several other solutions have been proposed for datacenter like environments, such as the PortLand [5], Al-fares

et al. [12], BCube [13], DCell [14], SPAIN [15], and the Virtual Layer 2 (VL2) [16]. All of these designs use varying degrees of topology constraints and end-host modifications to simplify the control and forwarding plane functionality. DBridges can be deployed in an arbitrary topology with RBridges or STP-based switches while retaining the plug-and-play feature of Ethernet. DBridges set no constraints on deployment strategy.

PortLand, VL2, and Monsoon implement a generalized directory service to reduce (ARP and DHCP) broadcasting in the network, which directly allows the broadcast domain size to be increased. PortLand uses a centralized directory service, while VL2 uses a distributed and hierarchical solution for scaling the directory service to larger Ethernet domains. Monsoon replaces ARP in the host network stack with a custom address resolution protocol, which breaks legacy application compatibility in the network.

3. Routing bridges

TRILL is a new Ethernet frame forwarding protocol standardized by the IETF. The devices running the protocol are called Routing Bridges (RBridges). RBridges are fully autoconfigurable, and retain the Ethernet plug-and-play nature of hosts. RBridges are incrementally deployable with spanning tree based Ethernet networks, and require no changes to hosts. Introducing RBridges in an STP-based Ethernet network will segment it into several smaller isolated Ethernet networks, each with its own spanning tree. In contrast to STP-based solutions, RBridges forward Ethernet frames using shortest path forwarding that is computed using a modified version of the well-known IS-IS link state protocol [17]. The forwarding operation is based on an encapsulation scheme that is added and removed at the edges of the network.

Fig. 1 presents the forwarding operation for Ethernet frames entering and exiting the RBridge network (a TRILL campus). When *Host A* sends a frame to *Host B*, it first traverses a normal Ethernet network. Upon reaching the network edge, an *ingress RBridge* RB_1 receives the *native Ethernet frame*, encapsulates it as a *TRILL Data frame*, and forwards it to the next hop on the path towards the *egress RBridge* RB_3. The ingress RBridge also performs conventional MAC learning on Host A during the encapsulation process.

As the frame traverses the TRILL campus, it goes through a number of *transit RBridges*, that forward the frame to the next hop on the path towards the egress RBridge, using the information contained in the encapsulation header. More specifically, the identifier of the egress RBridge included in the encapsulation header is used to index a forwarding table in the RBridge. The forwarding table contains the next hop MAC address on the path towards the egress RBridge, calculated by the shortest path first algorithm.

Upon reaching the egress RBridge RB_3, the TRILL Data frame will be decapsulated, leaving the native Ethernet frame intact. The frame is then emitted on the port where Host B is located. The egress RBridge also learns during the decapsulation process that Host A is located behind the ingress RBridge RB_1, recording an entry for the MAC address of Host A, and the identifier of the ingress RBridge in the local MAC learning database.

3.1. Control plane

RBridges discover the network topology and perform network autoconfiguration using a modified version of the IS-IS link state protocol. The link state protocol has a dual role in the network. First, it is used to negotiate link local features between neighboring RBridges to ensure host connectivity. Secondly, it is also used to advertise RBridge features and neighbor information across the TRILL campus.

Each RBridge in the TRILL campus is identified by two separate identifiers. Control plane functionality is driven by the IS-IS system identifier assigned for each RBridge, while the forwarding plane uses a separate identifier concept called a nickname. Each RBridge in the TRILL campus uses one or more nicknames. A nickname is a 16 bit quantity that is used in the encapsulation header to identify the ingress RBridge, the egress RBridge, or the multicast forwarding tree root. The link state protocol is responsible for automatically resolving any possible nickname collisions in the TRILL campus by announcing the used nicknames in link state protocol messages that are flooded throughout the campus.

3.2. Forwarding plane

Each TRILL Data frame traversing the TRILL campus is encapsulated with the format presented in the upper half

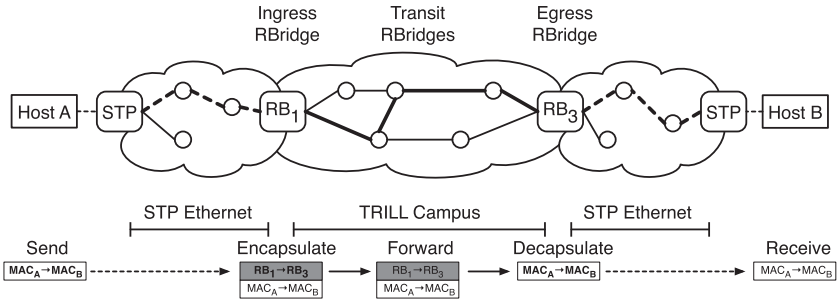

Fig. 1. The TRILL encapsulation and forwarding scheme.

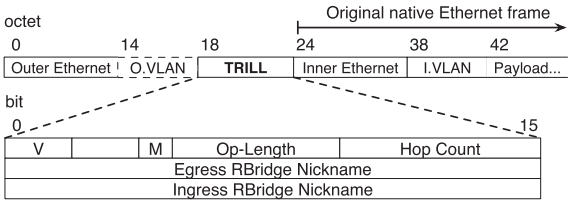

Fig. 2. The TRILL encapsulation format.

of Fig. 2 as octets beginning from the start of the Ethernet frame. The encapsulation process creates TRILL Data frames by inserting a separate Ethernet header (Outer Ethernet), and a special TRILL header in front of the native Ethernet frame (Inner Ethernet, I. VLAN, Payload, etc.). If VLAN tagging is used on the port, the frame also includes an outer VLAN tag (O. VLAN).

The lower part of Fig. 2 also includes an expanded view of the TRILL header, represented in bits. It is six bytes at minimum, containing enough information to forward the frame through the TRILL campus, and to guarantee loop safety within the campus via a hop count field. During the forwarding operation, RBridges modify the source and destination address fields in the outer Ethernet header to properly address the frames to the next hop on the path to the egress RBridge, and decrement the hop count field.

The forwarding operation uses two separate databases to send native Ethernet frames between hosts in the network: the MAC learning table, and the forwarding table. The MAC learning table is used to keep track of host locations in the network, while the forwarding table contains the shortest paths to all other RBridges in the TRILL campus.

The MAC learning table is functionally identical to the conventional MAC learning table found in Ethernet switches, however the location of a host in the network can be of two types. First, a host can be directly attached to an RBridge through one of its ports. In this case, the location is recorded as a port identifier. Secondly, a host may be located behind another RBridge in the network. In this case, the location is recorded as the nickname of the remote RBridge. The forwarding table is used to forward TRILL Data frames towards the egress RBridge. Conceptually, it contains an entry for each RBridge nickname in the TRILL campus, and the next hop on the path to that device.

RBridges retain the Ethernet plug-and-play model of communication, i.e., if a destination address is unknown (i.e., not found in the MAC Learning database) during the encapsulation process, the TRILL Data frame will be flooded throughout the network using a multicast forwarding tree.

3.3. Scalability

The intent of RBridges is not to affect Ethernet scalability, however the use of a tunneling header in frame forwarding alleviates some of the scalability issues in spanning tree based networks. As frame forwarding in the TRILL campus is based on the nickname information of RBridges, transit RBridges only need to have a forwarding table to guarantee end to end connectivity for hosts. This is a key distinction in the forwarding process, as the forwarding table scales with the number of RBridges in the TRILL campus, compared to the number of hosts in the network for spanning tree based forwarding. The RBridges on the network edges still require a conventional MAC learning table, that scales with the number of active hosts in the network.

To reduce the flooding that results from unknown destination forwarding, RBridges may optionally run a separate protocol on top of the link state protocol to advertise locally attached hosts to the TRILL campus. The End Station Address Distribution Information (ESADI) protocol allows RBridges to periodically multicast host attachments in a VLAN segment to other RBridges participating in the ESADI instance for that VLAN. While ESADI can be used to disseminate host information across the Ethernet network, a proactive and VLAN-segmented multicast model may quickly lead to significant control plane overhead in large scale or complex network installations.

4. SEATTLE

SEATTLE introduces a novel way to eliminate some of the inherent scalability issues in spanning tree based Ethernet networks. Exchanging the conventional spanning tree protocol with a link state protocol, and introducing Distributed Hash Tables (DHT) to the Ethernet networks allows the system to eliminate three major sources of broadcast in enterprise networks: ARP, ND, and DHCP. In addition, the DHT functionality is also used to implement host mobility support, directly on the link layer.

4.1. Consistent Hashing

Consistent Hashing [18] was first introduced as an efficient method to load balance resource usage in a distributed system. Since then, the design has been adopted in a wide variety of systems, ranging from peer to peer applications to distributed storage systems. For the purposes of this paper, Consistent Hashing and DHT are used synonymously.

Conceptually, a DHT stores (*key*, *value*) pairs (data elements) in nodes that are on a circular key space. Each node represents a networked device, typically server or a host participating in the DHT system. Each node is assigned a position on the key space by using a hashing function on a sufficiently unique identifier of the device, such as a switch identifier, or a host MAC address. Similarly, data elements are assigned to nodes in a DHT through a two phase process. First, the position of the data element on the key space is computed by using a hashing function on the data element *key*. Once the position of the data element is known, a mapping function (such as the minimum distance between the data element and the preceding or following node) is used to assign the data element to a node.

Each node participating in the system knows a subset (a *view*) of all the nodes in the DHT. Data elements are deliv-

ered to their assigned nodes using an "overlay network" that forwards the data element to its destination through a number of nodes using the underlying network primitives [19]. Typically, each node knows of one or more next-hop neighbors on the key space, and a set of distant neighbors that can be used to optimize the delivery process. Concretely, the data element is delivered towards the correct node by sending it to the known "next-hop" neighbor on the key space via the underlying network protocol. The next-hop neighbor then checks whether it is responsible for storing the data element, or if it must forward the data element to its own next-hop neighbor.

4.2. One-hop Distributed Hash Tables

The following discussion concentrates on a specialization of the algorithm, where all participants in the DHT know all other participants with very high probability [20]. The complete view of the nodes in a key space allows participants to directly (e.g., by using "one-hop") index the correct node for a given *key*. Typically, one-hop DHT systems have lower lookup latency and lookup failure rates than multi-hop systems, while the lookup table size grows linearly with the number of nodes in the system.

The key change in SEATTLE is to replace the conventional spanning tree protocol with a link state protocol. This allows the switches to advertise their participation in the DHT processing throughout the network. In a converged link state, this allows each switch to have a consistent view of all the nodes participating in the DHT. SEATTLE switches store information about directly connected hosts as (*key*, *value*) tuples to the DHT. A consistent view of the DHT is computed by assigning each switch to a circular key space using a hashing function on the switch identifier. The switch that is responsible for storing the information (a *resolver switch*) is determined by using a mapping function that selects the closest switch on the key space in counter-clockwise direction.

SEATTLE stores two types of information about the hosts in the network. First, host location information is stored in the DHT as (MAC_{host}, ID_{S_i}) tuples. The ID_{S_i} represents the switch level identifier of the first hop device for MAC_{host}. This information is used by the switches to remove flooding from the network, when the destination MAC address is not found in the local MAC learning database. Secondly, switches also publish IPv4 addressing information of directly attached hosts to the DHT in the form of (IP_{host}, MAC_{host}) tuples. SEATTLE uses the addressing information to implement proxy ARP behavior, where broadcast ARP requests are answered by switches on behalf of the hosts.

Fig. 3 presents an example of the SEATTLE one-hop DHT scheme in a simple topology. In the figure, the circular DHT key space spanning $[0,\ldots,1]$ is overlaid on top of a topology that contains three switches S_1, S_2, and S_3, and three hosts *a*, *b*, *c*. The hosts are each directly attached to switches S_1, S_2, and S_3, respectively.

The DHT key space is first populated by the switches S_1, S_2, and S_3 using the link state protocol messages that they receive from each other. In the example, each switch assigns all switches $(S_i, i \in \{1,2,3\})$ on the circular key space

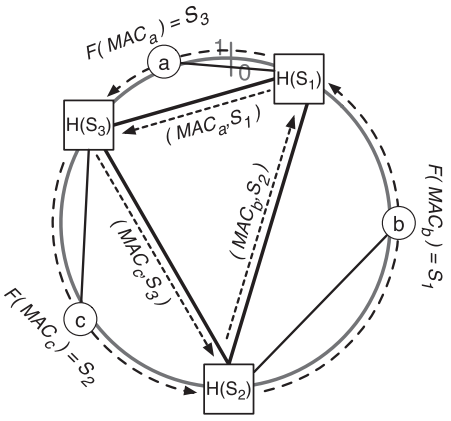

Fig. 3. Basic operation of SEATTLE one-hop DHT. Solid lines represent topology links, and the circular key space; dashed lines represent DHT signaling and metric function operation on the key space.

using a hash function $H(S_i) \in [0,\ldots,1]$ on the unique identifier of the switch. This operation is done by all switches in the network, resulting in an identical key space for all switches when the link state is converged. Each switch on the circular key space represents a node, where first hop switches store location and addressing information.

Fig. 3 also presents the operation of storing location information in the DHT. When switch S_1 receives an outgoing frame from its directly connected host *a*, it will update the location information of the host in the DHT. The switch responsible for storing the information is selected by using a mapping function $F(MAC_a)$ that hashes the MAC address of host *a*, and chooses the closest counter-clockwise switch (S_3) on the circular key space. Concretely, the information tuple (MAC_a, S_1) is delivered to switch S_3 using a unicast link state protocol message. The process is identical for the first-hop switches of hosts *b*, and *c*, however the resolver switches for the information are S_1, and S_2, respectively.

Similarly, IP addressing information of host *a* is stored on an *address resolution switch*, by using the IP address of the host as the key for the mapping function *F*. Note that host location, and addressing information may be mapped with a different function, and that $F(MAC_a)$ and $F(IP_a)$ may not map to the same resolver switch.

SEATTLE also automatically updates the DHT state during topology changes using Consistent Hashing. When a switch is added or removed from the topology, all other switches will automatically see the topology change from the link state messages. This allows the switches to recompute the resolvers for all information about directly connected hosts.

4.3. Host location and address resolution using DHT

When a SEATTLE switch S_a receives a frame sent to an unknown host, it will look up the resolver switch from the DHT using $F(MAC_{destination}) = S_r$. The frame will be delivered as unicast traffic to the resolver switch S_r, instead of being flooded throughout the network. Upon receiving the frame, the resolver S_r will forward it towards the des-

tination, based on the stored location information. As an optimization, S_r will also signal the location information of $MAC_{destination}$ to S_a.

SEATTLE switches also respond to broadcast ARP requests on behalf of the hosts in the network by redirecting the requests as unicast frames to address resolution switches. Addressing information can be learned from DHCP messaging, ARP requests and replies, or any IPv4 frame received from hosts. When an ARP frame requesting the MAC address of host d arrives at switch S_a, the switch looks up the resolver switch from the DHT using $F(IP_d) = S_v$. Upon receiving the unicast ARP request, S_v creates an ARP reply, based on the stored addressing information, and sends the reply back towards the requesting host. SEATTLE also optimizes the address resolution process, piggybacking location information of host d with the ARP reply.

Both host location and addressing information will be cached by SEATTLE switches. This reduces the amount of DHT signaling in the network, and the processing overhead in resolver switches.

5. DBridges

DHT Routing Bridges (DBridges) extend the base routing bridges standard with the one-hop DHT concept introduced in SEATTLE. The similarities between the SEATTLE architecture and routing bridges allow us to integrate the one-hop DHT scheme with minimal changes to the existing specification. The primary reason for this is the use of a link state protocol to discover the network topology, and to advertise device features across the network. In addition, both protocols also implement an encapsulation format that separates control plane traffic from user data frames. Finally, both protocols are also designed as "zero-configuration" protocols, requiring no administrative intervention to boot up the network in a default configuration.

DBridges use a simplified version of the one-hop DHT scheme from SEATTLE. More specifically, DBridges only cache location information, and leave addressing information uncached in the network. This compromise simplifies the address resolution process when IP addresses of hosts change, or when there is an address conflict in the network, while increasing the amount of address resolution related signaling and processing in the network.

The main difference between RBridges and SEATTLE is backwards compatibility. SEATTLE replaces all switches in an Ethernet network with new devices, while RBridges are incrementally deployable in networks with spanning tree based switches. To retain the incremental deployment property in DBridges, and by extension the Ethernet plug-and-play communication model, the SEATTLE one-hop DHT scheme has to be modified to support mixed networks with STP based switches, routing bridges, and DBridges.

We solve the incremental deployment problem by removing the strict requirement of SEATTLE one-hop DHT scheme, where all of the information about hosts in the network is always available in the DHT. The more realistic concept that the DHT does not hold all information has far reaching implications on the operation of DBridges, and allows us to retain all of the key features of both RBridges and SEATTLE while still supporting true Ethernet plug-and-play based communication. In turn, we introduce some processing and signaling overhead that presents itself in the network as flooding.

5.1. Control plane

Due to the similarities of the SEATTLE and RBridges control planes, the DHT advertisement mechanism fits directly into the IS-IS control plane operation. We have extended the advertisement protocol by introducing the concept of roles. A DBridge may act as a "client" in the DHT system, that only uses the DHT information to select resolver nodes for host location and addressing information. In contrast, a "server" in the DHT system can both use the DHT information to select resolver nodes, and store information provided by other DBridges through DHT signaling. This allows network administrators to exert tighter control over what DBridges in the network are responsible for information storage. Furthermore, allowing devices to only participate in the use of the DHT brings benefits on the edges of the network, where resource starvation (e.g., the capacity of the MAC learning table) on the devices may become an issue in large deployments.

5.2. Missing information in DHT

The core difference between the one-hop DHT scheme in SEATTLE and in DBridges is the method to deal with missing information in a resolver node. When an address or a host location frame is sent to a resolver node, the SEATTLE scheme presumes that the resolver node will have the information, and can complete the processing. If the information is not found on the resolver node, SEATTLE discards the frame, often leading to host service disruption. This is a reasonable presumption in enterprise networks, where administrators have better control over the devices in it. However, as can be seen from the list below, there are several transient cases where the information may not be found in the resolver, leading to host service disruption.

- The link state of the network has not yet converged,
- the resolver node suffers from resource starvation (e.g., no room to hold DHT information),
- hosts are either in bootstrap phase, or in pathological cases, receive only hosts, and
- in the case of DBridges in mixed networks, using the DHT to communicate with a host behind an RBridge.

We have relaxed the "omniscient" DHT requirement by reverting back to conventional Ethernet broadcasting for frames that have no related information in the DHT. This directly leads to increased processing overhead in the network, as we can no longer eliminate all frames that would otherwise be forwarded through the DHT system. However, it also allows us to retain the Ethernet plug-and-play model of communication in all situations.

Technically, whenever host address or location information is missing on a resolver, the DBridge simply returns the frame back to the ingress DBridge that sent the frame to the resolver. The frame is also modified by the resolver

DBridge to indicate that it is return traffic from a failed DHT operation. Once the ingress DBridge receives the frame, it changes the frame to a multi-destination TRILL Data frame and floods it to the network.

Fig. 4 presents an overview of the DHT miss operation in TRILL campus. When host *s* sends a frame to host *d*, the ingress DBridge DB_3 receives it. The MAC address of *d* is not found in the local learning database, so the DBridge computes the resolver of MAC_d with $F(MAC_d) = DB_2$, and encapsulates and forwards the frame. The resolver DBridge DB_2 receives the frame, but cannot find a location information entry for the host. It marks the encapsulated TRILL Data frame as return traffic from a failed DHT query, and forwards it as unicast back to the ingress DBridge. Upon receiving the frame, DB_3 re-encapsulates it as a multi-destination frame and floods it throughout the network. Finally, the flooded frame is received by the egress DBridge DB_1 that decapsulates it and forwards it to host *d*.

5.3. Incremental deployment with RBridges

DBridges are designed to be deployable in Ethernet networks with a mix of conventional spanning tree based switches and RBridges. Increasing the number of DBridges allows more host traffic to flow through the DHT, reducing the broadcast traffic in the network linearly.

When an ingress DBridge is forwarding a frame towards an unknown host in the network, it will automatically use the DHT and forward the frame through a resolver DBridge. If the host is behind an RBridge, the resolver will not have the location information. Thus, the frame will be returned to the ingress DBridge, that will flood the frame to the network to guarantee end to end connectivity between the hosts. As most flows between two hosts are bidirectional, the ingress DBridge will learn the location of the host behind an RBridge from subsequent message exchanges between the hosts through normal MAC learning process. After that, all communication between the hosts is done using the shortest path forwarding.

ARP requests are the prevalent source of the broadcast traffic in Ethernet networks. As addressing information is not cached at edge DBridges, each ARP request sent by a host connected to a DBridge is processed by an address resolver DBridge. Consequently, the address resolver receives requests for two kinds of addresses: hosts that are connected to DBridges, and hosts that are connected to RBridges.

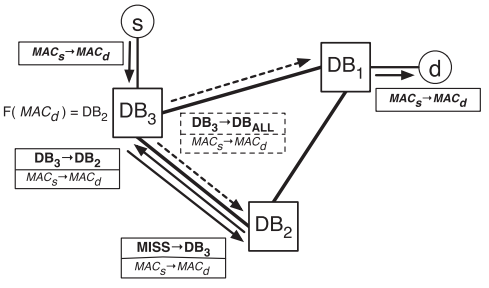

Fig. 4. DHT miss operation in DBridges.

Hosts connected to DBridges will have their addressing information stored at an address resolver DBridge. To optimize the latter case, we also allow DBridges to update information about hosts behind RBridges to the DHT. Each time a host behind a DBridge replies to a broadcast ARP request sent by a host behind an RBridge, we treat the destination addressing information of the ARP reply as a locally connected host. Concretely, this causes the DBridge to update the addressing information of the host to an address resolver DBridge on behalf of the RBridge. This allows the address resolver DBridges to reply to ARP requests targeting hosts behind RBridges, leaving only RBridge-based ARP request broadcasting active in mixed TRILL campuses. As a consequence, we also increase the DHT-related processing in both the address resolver nodes, and the edge DBridges.

5.4. Resource starvation in DBridges

Resource starvation in DBridges may occur due to two conditions in the network. First, the MAC learning database of a DBridge may fill up if the number of active hosts in the network exceeds the maximum size of the database. Secondly, if the DBridge acts as a resolver node in the TRILL campus, the node may receive more data elements to store than the maximum amount of storage space allocated for storing the DHT specific information. Both cases will cause a device overload situation, where new information is either not stored, or information is rapidly replaced in the DBridge.

If the MAC learning database of the DBridge fills up, adding new information into the database either becomes impossible, or causes an existing entry to be evicted. For the DBridge forwarding process, the resource starvation increases the number of frames that are sent through the location resolver node for the destination MAC address, however broadcasting does not increase in the network as a result of MAC learning database overload. DBridges prioritize the eviction of entries from the MAC learning table, so that the entries for remote hosts are removed before the entries of locally attached hosts.

If the storage space reserved for the data elements in the DHT fills up in resolver nodes, new information may be impossible to insert, or inserting it will cause an existing entry to be evicted from the DHT store. In both cases, resource starvation in resolver nodes reduces back to a case, where a part of the information is missing from the DHT. This in turn will increase the number of broadcast frames in the network. In the case of a location resolver store that is overloaded, the normal MAC learning process will reduce the number of broadcast frames that are sent to the network. For address resolution, each ARP request that cannot be responded by the overloaded address resolver node will revert back to the fallback broadcast operation presented in Section 5.2.

6. Software implementation

Our proof-of-concept implementation uses two independent processes to perform a subset of the RBridges base specification, and our DBridges extensions. First, the

RBridges control plane is run in the operating system as a user process, based on the popular Quagga routing suite [21]. Secondly, the forwarding plane is implemented with the Click modular router [22], allowing us to flexibly run the forwarding plane as a part of the operating system kernel, or as a user space process.

6.1. Click modular router

Click router configurations are built from individual components, called elements. To communicate with other elements in the router, each element may contain a number of input and output ports that receive or emit packets. Additionally, each port has a communication method it uses to receive or emit packets. A port operating with pull method will request packets from an upstream input port, process the packet and pass the packet towards the pulling element in the processing chain. In contrast, a port operating in push mode will process received packets and "push" the packets forward (downstream) in the processing chain through an output port.

Click elements can be combined in arbitrary order to form a directed, acyclic graph. A simple router configuration is presented in Fig. 5. It contains a *FromDevice* element, assigned to a network interface, a *Queue* element for storing frames generated by the FromDevice element, and a *ToDevice* element for emitting the frames out from a network interface. On the input processing chain, FromDevice is a pushing element, that outputs the packets downstream to the Queue element, that stores them. On the output processing chain, ToDevice is a pulling element, that requests the elements from an upstream input port, connected to the Queue and emits them on the network interface.

6.2. DBridges software architecture

Fig. 6 presents the overall architecture of our prototype. The control plane of the RBridges base specification is available in the Oracle OpenSolaris project [23] as a part of the Quagga package distributed with the operating system. We have modified the RBridges specific features to function outside of OpenSolaris, and implemented the necessary control plane extensions to support DBridges. The control plane is responsible for the TRILL link state protocol operation, i.e., it performs the necessary computations to allow the forwarding plane to correctly encapsulate, decapsulate and forward traffic in the network. The computed information is published to the forwarding plane using a standardized interface.

We implemented the forwarding plane of RBridges as a set of Click elements, dividing the frame processing logic into small independent functional blocks. The RBridge forwarding plane consists of approximately 3900 lines of C++ code, and the DBridge extensions add another 3600 lines of code, which includes all of the DHT related functionality. The forwarding plane consists of a set of frame processing elements that are combined into a chain that performs the complete input and output processing logic, and several shared elements.

Fig. 5. A simple example of a Click router configuration.

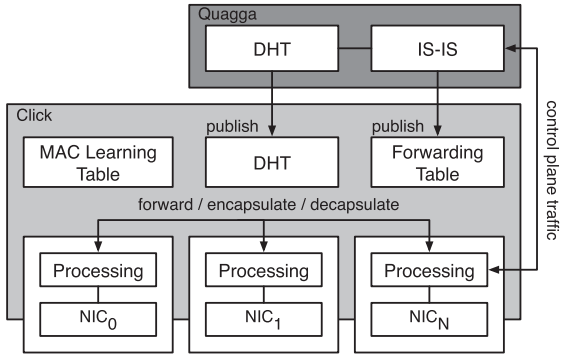

Fig. 6. Software architecture of the prototype.

Table 1
Throughput results for our implementation and the standard Linux kernel bridge.

	Frame size	
Implementation	64 Bytes	1518 Bytes
Software D/RBridge	2.57 Mfps	319 Kfps
Linux bridge	2.43 Mfps	325 Kfps

Each frame processing element is implemented in a way that allows "drop in" replacements to be installed in the router configuration. This allows us to isolate the DBridges related changes to individual elements, while keeping much of the forwarding plane frame processing identical between the two versions of the device. The division of work into individual elements also allows us to split some of the DBridges related processing (e.g., ARP and DHCP frame processing) into completely separate elements, that are simply added to the router through its configuration. An additional benefit of this separation is that we can forward the related traffic to the elements directly through the router configuration, instead of performing it inside the code.

The shared elements include the MAC learning table, the consistent hash ring (DHT), and the RBridges forwarding table. Each of these elements has a well defined service interface, that other (frame processing) elements in the forwarding plane can use. This was done to minimize the number of changes in the forwarding plane configuration when we transform an RBridge to a DBridge. As with the frame processing elements, a well defined service interface between the different versions of shared elements allows us to use them as "drop in" replacements in the forwarding configuration, or to test out new functionality or algorithms. In addition, we have designed the shared elements to minimize exclusive access in the service interface to reduce the overhead of locking in multi-cpu forwarding plane configurations.

6.3. Performance evaluation

While the implementation is a prototype version for research purposes, the system has been designed for parallel

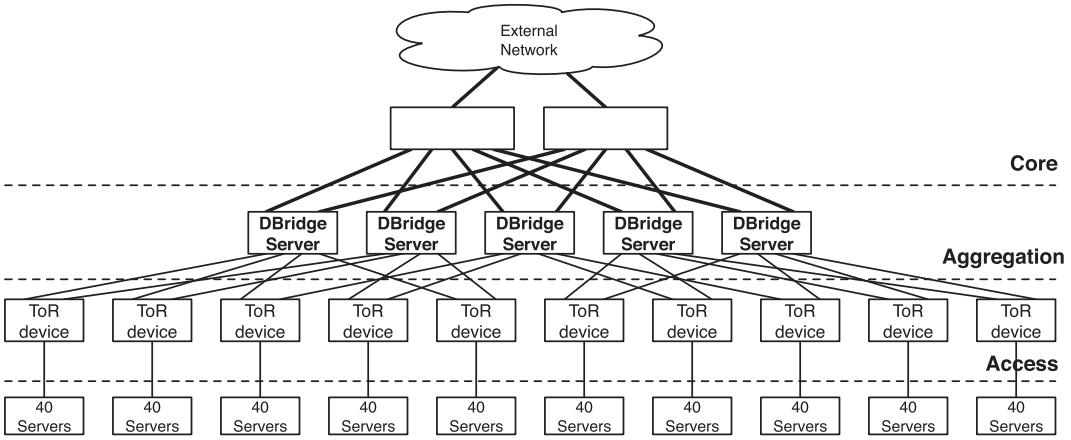

Fig. 7. The data center like topology used in our evaluation.

operation across modern multi-core processors for high-performance use cases. We have also previously measured the performance of our implementation using 1 Gbps ports [24], and compared its throughput characteristics against software based bridging in Linux.

As can be seen from Table 1, our implementation can match or exceed the offered throughput of the standard Linux bridge module in a test setup using four 1 Gbps ports. The slightly lower throughput for our implementation with 1518 byte frames is due to the extra bytes required by the TRILL encapsulation scheme, when both setups reach full line rate. In the future, we plan to test our implementation with modern 10 Gbps multi-queue network interface cards to fully take advantage of the modular design.

7. Evaluation

Our experimental evaluation compares DBridges to standard RBridges. Software implementations of RBridges and DBridges inside virtual machines are used to emulate a data center aggregation network, and a simplified version of the EBONE topology from Rocketfuel [25]. Furthermore, we also leverage the ns-3 [26] simulator and its real-time simulation features on the edges of the emulated networks to generate real Ethernet traffic in the network.[1] Two different traffic models are evaluated.

7.1. Test environment and goals

Our data center topology, presented in Fig. 7, is a simple three tiered version of the "canonical" data center layered infrastructure design. The network is divided into core, aggregation (inter-rack) and access (intra-rack) layers with redundant paths. Aggregation layer typically contains many of the complex services of the data center. In our tests, the five aggregation layer devices act as DBridge servers, while the access layer contains only DBridge clients or RBridges.

To show that our solution also offers benefits in a more generic network topology, we use the EBONE topology information from Rocketfuel to create a more mesh-like example topology. The topology was simplified by merging several nodes for a city to a single node, combining to a total of 23 node topology. We selected the five highest degree nodes to act as the DBridge servers in the network, and assigned every DBridge to service an equal portion of the hosts used in the evaluation.

Our test environment is a server running virtual machines to create a test network. Fig. 8 presents a high level overview of the interconnections between the different components in the test environment. The host operating system on the server is running several virtual machines, each representing a single switching device in the network topology.

Inside each of the virtual machines, we run the forwarding plane and control plane implementations of the devices under test. Each virtual machine is connected to a number of other virtual machines through virtual interfaces (vif) that are connected through bridges in the host operating system to create the links in the virtual topology. The virtual machines together with the links between their virtual interfaces in the host operating system create our virtual topology.

The evaluation server is also running ns-3, that is used as a real-time simulator process, emulating *all servers* in our evaluation topology with full IP stack. The NS-3 process connects itself with all of the virtual interfaces on the edge of the topology, and assigns each emulated server to their respective virtual interface, depending on the position of the emulated server in the network. The emulated servers generate real traffic flows to communicate with other emulated servers in the virtual network by sending traffic through the assigned virtual interface.

The goals of the evaluation are twofold. First, we want to verify that the inclusion of the SEATTLE one-hop DHT scheme does not adversely affect the operation of routing bridges, while retaining the benefits of the floodless communication model. Secondly, we want to verify that our extensions to the DHT scheme do not break existing behavior or eliminate its benefits, while allowing us to support incremental deployment and

[1] Note that ns-3 is only used for traffic generation and not for network simulation. The evaluation is done with a real, albeit virtual, test network using our proof-of-concept implementation.

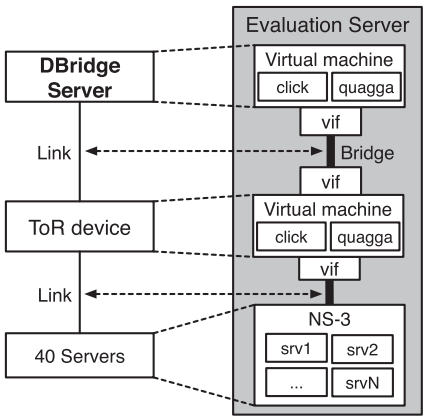

Fig. 8. Virtual test environment architecture.

certain transient network failures relating to resource starvation in switches.

To that end, we evaluate the effect of the SEATTLE DHT scheme inside a TRILL campus by measuring the amount of broadcast and signaling traffic, and comparing the results to standard routing bridges in identical environment. We also evaluate the effect of MAC learning table parameters (e.g., size, timeout) on the broadcast and signaling traffic in the network. In addition, we also evaluate our extensions to the DHT scheme by measuring the effect of incremental deployment and switch resource starvation on the amount of broadcast and signaling traffic in the network.

7.2. Traffic characterization

Recent efforts [27–29] that quantify traffic characteristics in various types of data centers allowed us to build two realistic traffic models that we used to evaluate our solution. We identified some common aspects from the research that we used as a basis for the flows we modeled inside our topologies. Most flows are small (<100 KB) and last a relatively short time (<10 s). Packet size is a heavily bimodal distribution, with clusters around 100 and 1500 Bytes. Flow interarrival times at switches and hosts are characterized by heavy-tailed distributions.

Our first use case loosely models a data center with distributed computing style applications (e.g. MapReduce [30]), where traffic flowing between the servers is mostly confined within a rack. However when the traffic gets out of the rack, it is directed at a large number of servers in other racks. The example EBONE topology also uses this traffic model to evaluate the design in a more generic network. In this case, our intent is to roughly model the flow initiation behavior of peer-to-peer applications in a hypothetical "large Ethernet" access network.

The second case models the data center of a hosting company, where user-facing applications (such as web servers) generate the majority of the traffic. Here, most of the traffic flows through the core layer to the external network and a small amount of traffic flows between independent server pairs.

Both topologies have a total of 400 emulated hosts in the network, split evenly between all switches that have end host service enabled. In the data center topology, there are 10 access layer switches for a total of 40 hosts per switch, and in the EBONE case there are 23 nodes for a total of 17 or 18 hosts per switch.

7.2.1. Peer-to-peer traffic model

The first traffic model loosely resembles a traffic matrix with distributed computing or peer-to-peer applications, with only intra- and inter-switch traffic. Tasks are created to model incoming requests at a host, which then connects to other hosts in the network (as actual traffic flows). The tasks are generated using a lognormal distribution, simulating a heavy tailed interarrival rate of new requests. We chose $lognormal(\mu = 2.0, \sigma = 1.0)$ and $lognormal(\mu = 3.0, \sigma = 1.0)$ distributions to represent the intra- and inter-switch task interarrival rate (in seconds).

In addition, based on [27], an exponential distribution is used to model the number of individual traffic flows in a single intra- and inter-switch task. For the intra-switch traffic, we defined the last percent of the probability density function as a case for "contacts nearly all other servers in the same rack", where the number of destinations is selected as a uniform random distribution of 90% to all but one server from the same switch. The number of flows for the intra- and inter-switch tasks are characterized by $exp(mean = 2.17)$ and $exp(mean = 8.695)$ distributions, respectively.

7.2.2. Web service traffic model

The second case models a traffic matrix with user-facing applications (e.g., web services). This traffic model is only evaluated using the data center topology. We add a single link to an unspecified "external network" from each of the core switches. These links are used to feed incoming web service requests to 10% of the hosts (web servers) on each of the ToR switches. The destinations for the requests are selected from the set of web servers using a uniform random distribution. The interarrival rate for the requests is modeled as an exponential distribution with a mean of 100 ms. This should adequately model Internet-facing traffic on web servers, as observed in [31,32].

The remaining 90% of the hosts are modeled as dependency services for the web servers. Each web server initiates new connections to the dependency services in their own ToR switch using the same distribution as the external links use for requests to the web services. The mean of the distribution is scaled up, proportional to the portion of the web servers in the full topology.

In addition, to model generic services used by all hosts in the system, (e.g. file or authentication services), we also add inter-rack connections from all the hosts, targeting any non-web service host in the racks. The initiation rate of these connections is again modeled with an exponential distribution, using a mean that is proportional to the interarrival rate of external requests to the whole topology, scaled up by 1.25.

8. Evaluation results

This section presents our results on how DBridges change the control traffic by measuring the amount of

frames per seconds in the switches. We also analyze how DBridges affect the broadcast domain size, the scaling of MAC tables, and incremental deployment. We also analyze the resiliency of our design.

8.1. Broadcast measurements

Figs. 9 and 10 present the network load of first hop RBridge and DBridge broadcasting, and DHT signaling in frames per second for the peer-to-peer traffic model in fully deployed RBridge and DBridge networks. We use DHT signaling to describe the unicast Ethernet frames in the network that DBridges use to manage the information content stored on the DHT servers. The web service traffic model shows similar traffic behavior. The broadcast traffic for RBridges consists of ARP requests that are flooded throughout the network, while the DBridge broadcasting is a result of DBridge server lacking the requested IP →MAC mapping. Note that in the figures, the small amount of broadcast traffic that occurs with DBridges is a result of the bootstrapping phase of our test, where the MAC tables of the network are empty, and hosts are being automatically discovered by the system.

The figures also show that DBridges eliminate most of the control traffic (TRILL encapsulated broadcasts and DHT signaling) from the network core, independent of the topology. Averaged over the whole test duration, the control traffic in a fully deployed DBridge network reduces by over 99.5% compared to a fully deployed RBridge network in both topologies. After the short bootstrap phase (20 s), we completely eliminate the largest source of broadcast traffic in the core, i.e., ARP requests.

RBridges generate a significant amount of TRILL encapsulated broadcasting in both topologies for the whole duration of the test. This is caused by the ARP operation, that requires both ends to learn the host IP →MAC mappings. In addition, ns-3 refreshes all active IP →MAC mappings every 120 s, causing a new ARP request/reply exchange for each of the hosts. The flow of ARP requests stays high throughout the tests because initially, all hosts will connect to destinations that cause an ARP request/reply exchange, and later on in the test, the concurrent active flows from all hosts creates a significant number of refresh ARP exchanges.

The locations and IP →MAC mappings for hosts are discovered quickly, as every connection to a new destination begins with an ARP request that can be used to extract the information. The bootstrap phase for DBridges in both topologies only lasts approximately 20 s, after which all of the host information can be found from the DHT and DBridge broadcasting is no longer used. The momentary peak during the bootstrap phase for the fallback broadcast is below 1000 frames per second in both topologies.

The DHT signaling adds minimal load into the network during the bootstrap phase and also periodically to refresh the stored information in the DHT. The signaling load reaches as high as 200 frames per second during the bootstrap phase but quickly drops to below 100 frames per second. Eventually, as all ingress DBridges have the location information for the hosts in the network, the signaling stops completely. The host location and IP →MAC mapping information for active hosts in the DHT is also refreshed using deferred messaging every 200 s. The duration of the DHT signaling during the bootstrapping of the EBONE topology is also roughly directly proportional to the number of switches that have directly attached hosts. This can be seen in Fig. 10 as a longer period (approximately double) of constant DHT signaling, when compared to the duration of the signaling in the data center topology.

Fig. 11 presents the network-wide control traffic for the peer-to-peer traffic model in both topologies when the number of active hosts in the network increases. The number of ARP requests increases dramatically for both topologies while the growth of the DHT signaling is significantly more gradual. Our rough estimation is that the broadcast domain of a DBridge network can be at least ten times larger compared to an RBridge network before DHT related signaling is noticeable in the network.

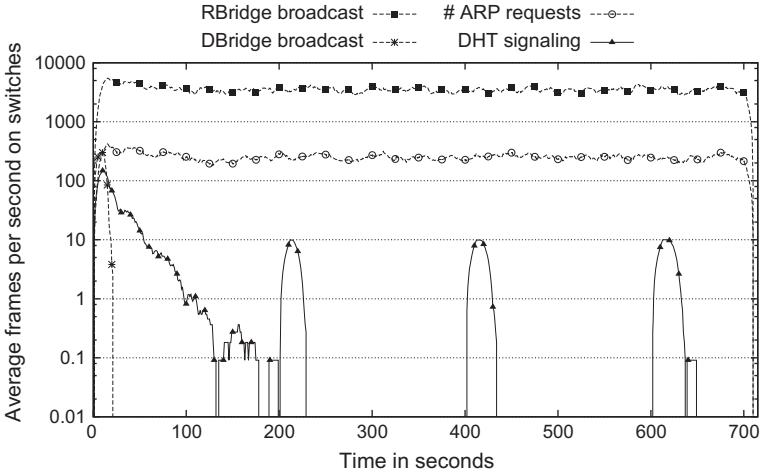

Fig. 9. Packet type load for RBridges, DBridges, and ARP first hop broadcast in the data center topology. Log scale.

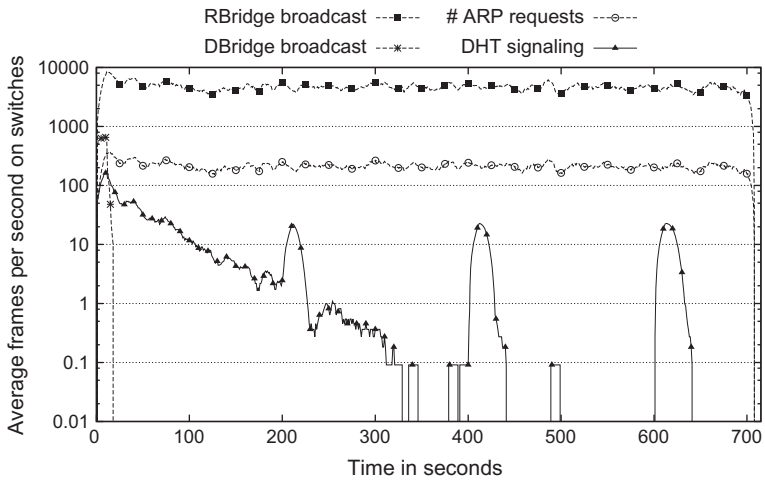

Fig. 10. Packet type load for RBridges, DBridges, and ARP first hop broadcast in the EBONE topology. Log scale.

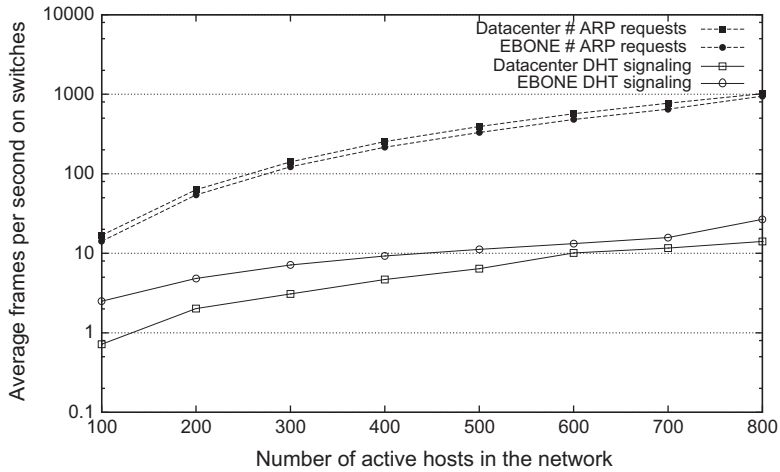

Fig. 11. Scalability of DBridge features in broadcast domains with the peer-to-peer traffic model. Log scale.

8.2. MAC table size

As unknown destination frames are no longer broadcast in fully deployed DBridge networks during normal operation, unused MAC table entries can be expired aggressively on first hop DBridges. The entries for the locally attached hosts use a normal MAC table timeout in the hundreds of seconds, but the location information of distant hosts can be removed sooner, depending on the traffic characteristics of the network. At DBridge servers, the stored information is held for a longer time, at minimum the duration that the first hop DBridges use for locally attached hosts.

Our solution allows fully deployed DBridge networks to lower the MAC table timeout value for distant hosts. Timeouts as low as 10 s are feasible, allowing networks to serve roughly 50% more active hosts without significant traffic or processing overhead on DBridges. Coupled with the granularity of the TRILL IS-IS DHT participation mechanism and the behavior of DBridges with MAC table overloads, this should allow edge DBridges to function with lower memory and processing requirements than edge RBridges.

Fig. 12 presents the average MAC learning table fill level and location query frequency for edge DBridges with various time out values for the MAC table distant host entries. The fill level is defined as the ratio between the average number of entries in the MAC tables of the edge devices and the total number of unique host MAC addresses that a DBridge can see during the test. The figure presents the fill level and location query frequency of (1) data center and EBONE topology edge DBridges in the peer-to-peer traffic model, and (2) data center topology edge DBridges in web service traffic model. A location query occurs when the MAC address of the destination host is missing from the local MAC table.

Both traffic models exhibit similar behavior in the data center topology when the MAC table timeout value for distant hosts is increased. The behavior is related to how Ethernet MAC learning tables function. When a two-way connection is initiated between hosts, both ends of the connection are learned by the edge switches on the path. The location of a distant host is removed from an edge switch if the distant host does not communicate with *any*

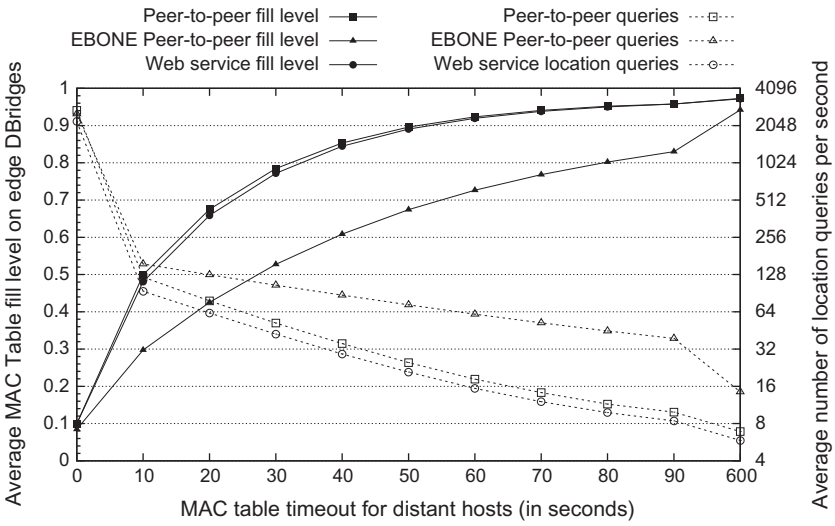

Fig. 12. Average MAC table fill level and DHT location queries in access layer DBridges. Right y-axis in $\log_2$ scale.

of the hosts behind the switch during the distant host MAC table timeout interval.

The EBONE topology has a lower average fill level, and a higher number of location queries than the data center topology. This is a direct result from the fact that the EBONE topology edge DBridges have less than half the number of attached hosts, compared to the data center edge switches. This directly affects the traffic model used to generate tasks and traffic flows in the network. Consequently, the average fill level is lower, because there are fewer communicating host pairs per DBridge. This also increases the number of location queries in the network, as the host location and IP → MAC mapping information is propagated more gradually to the edge switches in the network.

We also measured two special cases for the MAC tables: no caching (timeout of 0 s), and no changes (timeout of 600 s) to timeout values of distant host locations. With no caching, ingress DBridges send all distant host traffic through a DBridge server, and the MAC table fill level stays constant, i.e., the amount of attached hosts to each of the edge DBridges. When the distant host location information has no separate timeout value, we can see that the MAC tables in all of the edge DBridges are practically full in all topologies and traffic models.

With no caching, the amount of MAC table misses on the edge DBridges is between 2200 and 2700 misses per second in all topologies and traffic models. In an ideal case, this amounts to 500 DHT operations per second on each of our DBridge servers. The number of operations in this case is directly proportional to the amount of traffic in the network. As we were only interested in broadcast minimization, the numbers presented here are artificially low. In most real networks, it is likely that disabling distant host location learning on edge DBridges is not feasible due to the processing overhead it will cause on the DBridge servers in the network.

8.3. MAC database overload

Increasing the number of active hosts (unique MAC addresses) over the MAC table size creates MAC table misses on the edge of the Ethernet network, as not all host location information can fit in the table. In our RBridge and DBridge implementations, location information is still learned from received TRILL encapsulated frames in overload situations. This causes a random distant host location entry to be removed from the MAC table. A table miss in RBridges causes the frame to be flooded throughout the network to ensure that it reaches the destination. DBridges will use the DHT to forward the frame as unicast to the DBridge server for the destination host MAC address.

Fig. 13 presents the results for a test where the MAC table size in edge devices was bound to a maximum of 300 entries, but the number of active hosts in the network gradually increases from 260 to 440. At 440 active hosts, each switch MAC table on the edge of the network is overloaded by 47% (i.e., $\frac{(440-44)-(300-30)}{(300-30)}$). After 300 active hosts, the MAC tables on the network edge begin experiencing table misses. As the MAC tables fill up, RBridges begin flooding the unknown destination frames throughout the network. DBridges replace flooding with unicast forwarding through a DBridge server. The operation also causes the location of the destination host to be signaled back to the DBridge client using. We can also see that the fallback broadcasting used by DBridges scales independent of the MAC table size.

Fig. 14 presents an estimate of the projected scaling of the relevant control traffic, when the number of active hosts in the network is increased up to 1000 hosts, while the MAC table size of the switches is 300 entries. We can also see that the DHT signaling traffic in the network and ARP requests sent by hosts scale only gradually, while the RBridge based flooding due to host ARP requests increases dramatically as the number of active hosts grow.

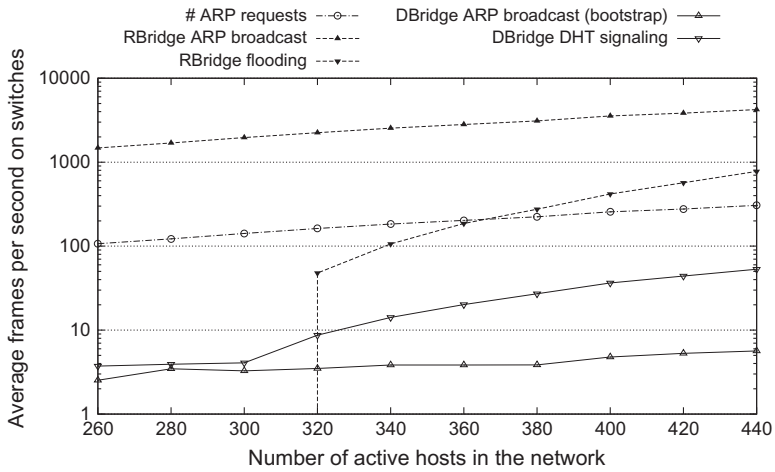

Fig. 13. Control traffic increase in fully deployed RBridge and DBridge data center network with a MAC table size of 300 entries.

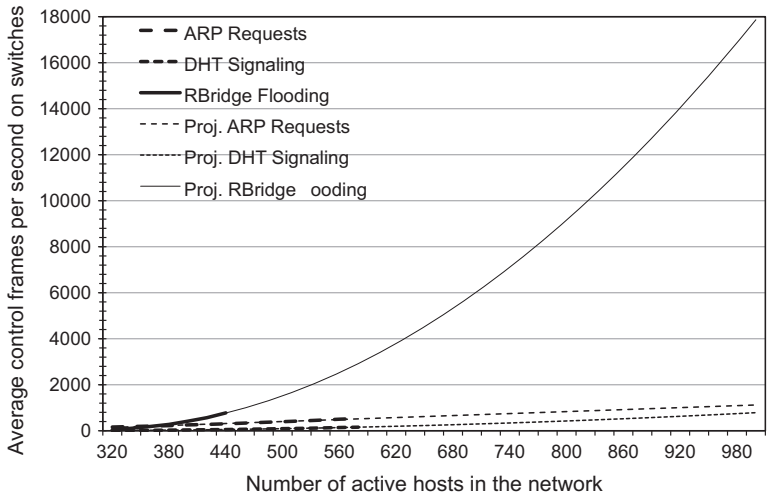

Fig. 14. Control traffic scalability of DBridges and RBridges in fully deployed data center network with a MAC table size of 300 entries.

Overall, we can see an order of magnitude difference in control traffic between RBridges and DBridges. Another benefit of the DHT signaling scheme is that the traffic is sent as unicast, allowing shortest path forwarding and multipathing, while RBridge flooding is delivered using one or more network-wide distribution trees.

8.4. Incremental deployment

Fig. 15 shows the reduction to control traffic in the network, when the number of DBridges in the network increases. In both topologies, we replace random RBridges in the network that have any attached hosts. Control traffic consists of broadcast TRILL Data frames, either by RBridges or DBridges due to a fallback mechanism, and the small amount of DHT signaling used by DBridges. Each added DBridge brings another set of hosts into the DHT that increases the likelihood that a pair from the set of all hosts in the network are connected to DBridges. Overall, the figure shows a large reduction (two orders of magnitude) in broadcast traffic, when all hosts are connected to DBridges. In this case, the only broadcast traffic in our tests is the brief period of fallback broadcasting at the beginning of the test. What is also notable is that while the broadcast traffic is clearly affected by the traffic volume and topology complexity, the DHT signaling in the network is insignificant during the incremental deployment.

8.5. DHT resiliency

Figure Fig. 16 presents the control traffic load in the network when a DBridge (one of the aggregation layer DBridges in the data center topology) crashes 300 s into the peer-to-peer traffic model test. The total convergence time (approximately 30 s) for the TRILL IS-IS link state protocol is highlighted with gray in the figure. Note that the relatively long period of link state convergence is a result of the TRILL protocol.

In the figure, we can see a momentary burst of DHT signaling frames emitted by the ingress DBridges as they con-

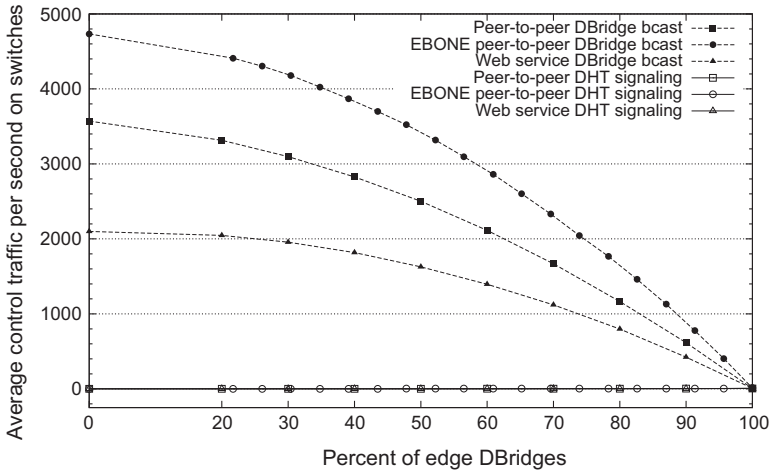

Fig. 15. Effect of incremental deployment of DBridge nodes on control traffic.

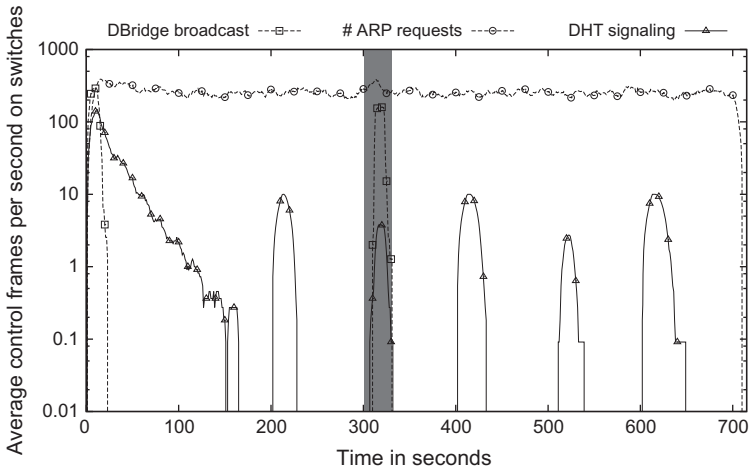

Fig. 16. Control traffic load during a crash. Link state convergence time highlighted in gray. Log scale.

verge to the new DHT state and perform a rehash of the location and addressing information of directly attached hosts. As a result, the periodic refreshing of the information in the DHT is also split into two separate periods that are seen in the figure as the test progresses.

Additionally, the figure also shows that during the convergence period, DBridges use fallback broadcasting to deliver frames to hosts. In the case where an ingress DBridge has already converged to the new state, and the DBridge server has not yet received the information, the ingress node will revert to fallback broadcasting. This guarantees that end host service still functions, while the DHT state is gradually corrected as the updated link state information propagates throughout the network.

The topology change also causes a brief service disruption (not seen in the figure) due to forwarding table changes and DHT state inconsistencies. Host service disruption caused by our DHT extension happens when an ingress DBridge attempts to use a DBridge server that has not yet converged to the new network state, and the information is not found. Consequently, the server will not return the frame to the ingress node for fallback broadcasting because it does not consider itself to be the server for the information, i.e., the IP → MAC mapping for ARP requests, or the host location information for a MAC address.

9. Conclusions

Extending RBridges with the SEATTLE one-hop DHT scheme offers tremendous benefits in arbitrary Ethernet topologies. Furthermore, our extension to the DHT scheme enables the network to function in transient failure cases and during switch resource starvation without severely increasing the processing and traffic overhead in the network. DBridges effectively remove the major sources of broadcast from Ethernet networks while retaining backwards compatibility with standard IETF-based RBridges. Finally, DBridges require no changes to hosts, preserving the vision of a plug-and-play Ethernet.

Security features and the mobility of virtual and physical end hosts are improved, although we leave the details and

analysis as future work. We will also investigate extending the one-hop DHT design to additional broadcast-driven protocols. DBridges are a very promising solution that can be deployed gradually in existing Ethernet networks.

References

[1] Media access control (MAC) bridges, Standard 802.1D-2004, IEEE Computer Society.
[2] D. Plummer, Ethernet address resolution protocol, RFC 826, IETF, November 1982.
[3] T. Narten, E. Nordmark, W. Simpson, H. Soliman, Neighbor discovery for ip version 6 (IPv6), RFC 4861, IETF, September 2007.
[4] R. Droms, Dynamic host configuration protocol, RFC 2131, IETF, March 1997.
[5] 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: SIGCOMM, ACM, 2009, pp. 39–50.
[6] R. J. Perlman, Rbridges: transparent routing, in: INFOCOM, 2004, pp. 1211–1218.
[7] R. Perlman, D. Eastlake, D. Dutt, S. Gai, A.Ghanwani, Routing bridges (rbridges): base protocol specification, RFC 6325, IETF, March 2010.
[8] C. Kim, M. Caesar, J. Rexford, Floodless in SEATTLE: a scalable ethernet architecture for large enterprises, in: SIGCOMM, ACM, 2008, pp. 3–14.
[9] A. Myers, T.E. Ng, H. Zhang, Rethinking the service model: scaling ethernet to a million nodes, in: HotNets, ACM, 2004.
[10] T.L. Rodeheffer, C.A. Thekkath, D.C. Anderson, SmartBridge: a scalable bridge architecture, in: SIGCOMM, ACM, 2000, pp. 205–216.
[11] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, ROFL: routing on flat labels, in: SIGCOMM, ACM, 2006, pp. 363–374.
[12] M. Al-Fares, A. Loukissas, A. Vahdat, A. scalable, commodity data center network architecture, in: SIGCOMM, ACM, 2008, pp. 63–74.
[13] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: a high performance, server-centric network architecture for modular data centers, in: SIGCOMM, ACM, 2009, pp. 63–74.
[14] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, S. Lu, DCell: a scalable and fault-tolerant network structure for data centers, in: SIGCOMM, ACM, 2008, pp. 75–86.
[15] J. Mudigonda, P. Yalagandula, M. Al-Fares, J.C. Mogul, SPAIN: cots data-center Ethernet for multipathing over arbitrary topologies, in: NSDI, USENIX, 2010, pp. 265–280.
[16] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: SIGCOMM, ACM, 2009, pp. 51–62.
[17] Intermediate system to intermediate system intra-domain routeing information exchange protocol, Standard 10589:2002, ISO/IEC, 2002.
[18] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, D. Lewin, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, in: STOC, ACM, 1997, pp. 654–663.
[19] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for Internet applications, in: SIGCOMM, ACM, 2001, pp. 149–160.
[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: Amazon's highly available key-value store, in: SOSP, ACM, 2007, pp. 205–220.
[21] The quagga routing suite, <http://www.quagga.net/>.
[22] E. Kohler, R. Morris, B. Chen, J. Jannotti, M.F. Kaashoek, The click modular router, ACM Trans. Comput. Syst. 18 (2000) 263–297.
[23] OpenSolaris rbridge (IETF TRILL) control plane support, <http://solaris.java.net/>.
[24] N. Varis, J. Manner, Performance of a software switch, in: HPSR, 2011, pp. 256–263.
[25] Rocketfuel maps and data, <http://www.cs.washington.edu/research/networking/rocketfuel/>.
[26] Network simulator 3 (ns-3), <http://www.nsnam.org>.
[27] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: measurements & analysis, in: IMC, ACM, 2009, pp. 202–208.
[28] T. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, SIGCOMM Comput. Commun. Rev. 40 (1) (2010) 92–99.
[29] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: IMC, ACM, 2010, pp. 267–280.
[30] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113.
[31] P. Barford, M. Crovella, Generating representative Web workloads for network and server performance evaluation, SIGMETRICS Perform. Eval. Rev. 26 (1998) 151–160.
[32] B.D. Allen, Lognormal and pareto distributions in the Internet, Comput. Commun. 28 (7) (2005) 790–801.

Nuutti Varis is a postgradutate student at Aalto University. He received his M.Sc. degree in 2008 from Helsinki University, specializing in distributed systems and data communications. His current research interests include current and next generation Ethernet technologies, software defined networking, and commodity hardware based network devices.

Jukka Manner (born 1972) received his M.Sc. (1999) and Ph.D. (2004) degrees in computer science from the University of Helsinki. He is a full professor (tenured) of networking technology at Aalto University, Department of Communications and Networking (Comnet). His research and teaching focuses on distributed systems and various networking aspects, most recently on the development of the Internet and it's services, particularly in topics related to energy efficient ICT, networking beyond IP, multipath connectivity and transport protocols. He is the Academic Coordinator for the Finnish Future Internet research programme. He is an active peer reviewer and member of various TPCs. He has contributed to standardization of Internet technologies in the IETF for over 10 years, and was the co-chair of the NSIS working group. He has been principal investigator and project manager for over 15 national and international research projects. He has authored over 80 publications, including several IETF RFCs. He is a member of the ACM and the IEEE.

Mikko Särelä works as a post-doctoral researcher at Aalto University. Prior to joining Aalto University, he worked at Nomadiclab, Ericsson on information centric network architecture and future Internet. His current research interests include ethernet scalability and security, energy efficiency, and mobility.

Timo Kiravuo (born 1965) is a postgradutate student in Aalto University. He received his M.Sc. (1999) from Helsinki University of Technology. After a career in private sector his work focuses on Internet security and related matters. Currently he is researching the security of Ethernet technologies.