A Secure Multicast Tree for the Global Data Plane

Arun Sundaresan
UC Berkeley
arun.sundaresan@berkeley.edu

Mikkel Svartveit
UC Berkeley
misva@berkeley.edu

Tony Hong
UC Berkeley
tshong@berkeley.edu

Abstract

We present a multicasting protocol for the Global Data Plane, capable of utilizing the entire topology of the network to build efficient multicast trees. Whereas earlier approaches simply route through a strict tree hierarchy of trust domains, our proposal enables the use of links between arbitrary domains at any level of the trust domain hierarchy. Our protocol dynamically builds a multicast tree by leveraging an end-to-end principle for trust domains, dividing the problem into cross-domain and intra-domain routing. We also present methods for multicast groups to ensure confidentiality and integrity over messages sent, while managing access revocation and ensuring only authorized clients receive multicast messages. We detail the implementation of a simulator for multicasting and demonstrate that our protocol achieves significant reductions in stress at the higher levels of the trust domain hierarchy. At the same time, we are building multicast trees that are more efficient relative to a strictly hierarchical protocol. We also show that the overhead of encrypting messages fades as more messages are sent to the same multicast group. Finally, we present methods for preserving multicast groups in cases where resources fail or deliberately leave the network.

1 Introduction and Related Work

1.1 The Global Data Plane

The Global Data Plane (GDP) is rooted in a trend towards moving more powerful computing resources towards end users. However, this trend does come with its own set of challenges. For instance, managing edge devices can involve detecting and resolving faults on thousands of different devices. The GDP provides a unified and secure method to manipulate resources on the edge and aims to support communication between edge devices via publish-subscribe messaging. Multicasting is particularly important to the GDP since limited network bandwidth is often a constraint in edge computing use cases, making alternatives like broadcasting too inefficient for practical use (Mor et al., 2019).

1.2 GDP Routing and Network Organization

GDP routing partitions nodes into *routing domains*. Each one has a *GLookup* service that stores routing information for all nodes within the routing domain. If a destination node cannot be found in the GLookup for the current routing domain, the sender can query the parent domain's GLookup (routing domains are organized hierarchically) (Mor et al., 2019). Unlike IP, the GDP cannot rely on methods like prefix routing since every entity in the GDP has a 256-bit GDP name (Inc., 2001).

1.2.1 Multicast Groups

In the same vein of thinking, we partition our clients into "multicast groups". A client can be a part of several multicast groups and there is no semantic meaning to a multicast group (for example, members of a multicast group may be spread across the world) other than the fact that all clients in the group should receive messages addressed to the 256-bit GDP name of said multicast group.

1.2.2 Trust Domains

Additionally, clients are members of a trust domain. Notably, each trust domain consists of a collection of physically close devices with common ownership (Mor et al., 2019). For example, the University of California may be a trust domain, with Berkeley as a child, and the Department of Electrical Engineering and Computer Sciences as a grandchild. The key idea for routing is that we want to maximize sending messages within trust domains and keep the path as short as possible (in practice, this means we only want to send messages wherever they are necessary; for example, a message within a trust domain does not need to go to the global GLookup). We assume that the network within a trust domain is dense with short links, while links between trust domains are sparser and with longer links. We also assume that more nodes exist than are needed to create a multicast tree among the given trust domains.

1.2.3 Routing Information Bases

Trust domains each have a Routing Information Base (RIB) that stores information about the GDP names that are members of a trust domain, how to route messages within the trust domain, and how to route between descendant trust domains. The idea of RIBs enable trust domains to organize themselves in a tree structure, so trust domains have a parent (except for the root) and some number of children. We say each trust domain has a *domain router* consisting of a RIB and an entity capable of sending messages to direct members of the trust domain, in addition to parent and child trust domains. Importantly, other GDP resources such as switches are also members of a trust domain.

1.2.4 PSL Multicast Paper

In 2021, Plutowski et al. proposed a multicast tree building protocol for Paranoid Stateful Lambdas (PSL) nodes where trust domains took the form of a n-ary tree (Plutowski et al., 2021). Moreover, they did not provide any security guarantees over their messages or address the idea that trust domains could be connected via links not part of a strict tree hierarchy. To this end, their approach puts undue stress on routers high in the hierarchy instead of leveraging the non-tree links. Our approach generalizes the assumptions made in the 2021 paper, attempting to provide utility for any application built on top of the GDP. We present a dynamic multicast tree building protocol that can leverage all links in the network while preserving a trust domain hierarchy, alleviate congestion at higher levels in the hierarchy, and provide end-to-end symmetric encryption of messages with capabilities for key rotation and access revocation.

2 Design

2.1 Terminology

The naming convention we use for devices within the network distinguishes between clients, switches and routers.

2.1.1 Clients

Clients represent devices in the network that produce and consume messages. They are likely personal computers or servers or some kind. A client is connected to a single switch or router, and all traffic from the client goes through this link.

2.1.2 Routers

Routers, as the name suggests, are responsible for doing the heavy lifting of message routing in the network. Routers are laid out in a hierarchy, which implies that each router has a designated parent router and a number of child routers. Each router defines a routing domain in the network. Despite the hierarchical nature of routers, we allow a router to have links to other routers that are not its parent or any of its children. As mentioned earlier, each router stores a Routing Information Base (RIB) that stores 1) information about all nodes owned by the domain and links between them, and 2) child domain routers and links between them. Note that a RIB does not store information about clients and switches within child domains.

2.1.3 Switches

Switches sit between routers and clients within a routing domain. They can have an arbitrary number of connections to clients, routers, and other switches. Switches are conceptually simpler devices than routers. They only keep a local routing table in the form of a key-value store, with GDP names as keys and a set of next hops as the corresponding value. If a switch receives a message with a destination name that is not stored in its local routing table, the switch will send a message to its domain router. The router will query its RIB and return a set of next hops to the switch. The switch caches this information in its local routing table, so that the routing information can be reused next time the switch encounters this GDP name.

2.2 Goals

We should use routing information bases at each level to determine what the next destination for a message is. Everything in the tree has a unique 256-bit GDP name, including the switches, clients, and routers. We want to find efficient paths for each message. Moreover, our multicast tree should be dynamically built and only use resources as close as possible to joining entities.

We want to provide end-to-end symmetric encryption for messages, having clients in a multicast group agree to one encryption key for the whole multicast tree. Furthermore, it should be straightforward to rotate keys and remove clients from the tree.

We aim to support tree construction and repair. Specifically, we propose a method to detect when neighboring elements of the tree fail, and figure out a way to replace them in a way that does not impact the functionality of existing multicast trees or trust domains.

2.3 Threat Model

We assume that routers and switches are honest-but-curious, but that switches only offer unreliable delivery. We assume that each client and router has a public/private key pair to be used for asymmetric encryption, another asymmetric pair to be used for signing and verification, and that the public keys are authorized by an external, trusted certificate authority. We are convinced that this is a sound threat model, as protocols for messaging between members of the GDP (such as DTLS) provide protection against an adversary who tries to modify packets in flight. Additionally, prior literature about privacy preservation has also taken an honest-but-curious adversary to be the main threat.

2.4 Trust Domains

Trust domains will be used to partition clients and handle routing, both inside and outside the domain. Each trust domain router has a RIB that keeps track of the clients and switches in the trust domain, the links between them, as well as which multicast groups those clients participate in. Additionally, RIBs store information about where to find parent and child trust domains. Each trust domain has a router that forwards messages and can leverage information from its corresponding RIB. All clients and switches are members of one and only one trust domain.

2.4.1 Creating trust domains

Trust domains are created by sending a message to the parent trust domain router, which checks the authorization of the node sending the message and may reject the request. If accepted, the node that originally sent the message must initialize a router (including a RIB) and supply the GDP names to the parent, which adds them to its RIB. The parent router propagates a message up the tree, forwarding to successive ancestors its information about the new domain. Since each domain router should have visibility of all resources within their trust domain, routers build an adjacency list of the resources available in their domains.

2.4.2 Joining Trust Domains

A domain router may add additional resources to its trust domain. To do this, the router must specify a 256-bit GDP name for the new resource and add it to its RIB. A similar procedure can be performed for adding new neighbors if new switches come into the graph. Additionally, we only want authorized clients to be able to join a trust domain. Hence, we require that clients conducting an operation (join/leave) on a trust domain must present a certificate (stored by a globally visible certificate authority) to be verified by the router of the trust domain to verify its identity. Additionally, trust domain routers wishing to establish connections with another trust domain router must also present their certificates.

2.4.3 Leaving Trust Domains

The router may also remove resources from its trust domain. This is done by removing the resource's GDP name from the RIB for clients that are direct members of the trust domain. For GDP switches, the process is more complicated. A switch leaving may break a multicast tree (this will be addressed in the fault tolerance section later).

2.4.4 Consistency

A problem that might arise is that updates to the network structure can arrive from multiple trust domains at once. For example, a request to create a multicast tree may arrive from trust domain A and use a switch owned by trust domain B, which is attempting to remove that switch. We discuss this problem in more detail when we examine the procedure of building a multicast tree.

2.5 Multicast Groups

Cross-trust domain routing involves flooding a multicast tree (potentially including nodes outside the source trust domain) with a message. Messages are sent from the source's trust domain router to other recipient trust domain routers. We assume trust domain routers are aware of child and parent trust domain routers and have some idea of the length of time a message will take to go from the source trust domain to a neighboring trust domain. We assume that edge lengths within a trust domain are short and edges are dense. Between trust domain routers, we assume edges are sparser and longer.

2.5.1 Creation

A client can be a member of several multicast groups (but exactly one trust domain) and there need not be any relationship between members of a multicast group (physical, logical, or otherwise). A client can form a multicast group by messaging the router of its trust domain.

Multicast Group Leaders Whenever a new multicast group is created, the first client becomes the "leader" of the group and is responsible for assigning and rotating keys in collaboration with its corresponding domain router (which emits multicast messages to send new keys). The leader client of a multicast group can specify new members to join the multicast group at any time by signing a message that the member's GDP name is allowed to join and sending these messages to a certificate authority. The leader sends these signed messages to the certificate authority, and the trust domain router of any client that seeks to join a multicast group verifies that such a message exists before proceeding further.

Setting Up Keys We want messages to be encrypted in transit and only decryptable by clients that are part of the appropriate multicast group. Thus, each multicast group must have separate encryption keys and a procedure to rotate them and revoke access. We assume that any client in the multicast group have a public-private key pair that can be used for an initial setup of keys used for encrypted messaging. When a multicast group is created, the leader generates separate symmetric keys for encryption and HMAC.

2.5.2 Joining

When a client wants to join an existing multicast group, it needs to establish a path between itself and the existing multicast tree. A key principle when building the multicast tree is to keep the procedure as close in the trust domain hierarchy as possible. To this end, the client starts by messaging the router of its trust domain. If the router of the client's trust domain is already part of the multicast tree, the router adds the client to its list of direct members that participate in the multicast group and computes a path to send messages to the client (similar to the procedure below). Otherwise, the router leverages the hierarchy of trust domains. This is when the multicast tree between trust domains is built.

Finding the Lowest Common Ancestor If the client's domain router is not already in the multicast group, the router asks its parent router if it has information about the multicast group. This process continues until some trust domain router has information (in these messages, we also note the original router that sent them, as well as the client wishing to join). We refer to this router as the *lowest common ancestor*. This procedure keeps multicast tree building as local as possible, confining it to a subtree rooted at the lowest common ancestor rather than involving all trust domain routers. At the lowest common ancestor, we compute a path from the client requesting to join to the rest of the multicast tree.

Path Computation Approaches We want to efficiently find a low-cost path between the client requesting to join and the rest of the multicast tree. To this end, we can require trust domain routers to maintain an adjacency list containing all the resources in the trust domain and well as within descendant trust domains. We can use a path-finding algorithm on the adjacency list in the lowest common ancestor. Starting at the joining client, we find the distance to the neighboring resources and get the minimum of (distance + distance to multicast tree) for each node. The case for DFS over dedicated shortest path-finding algorithms is supported by performance and simplicity. DFS runs in O(V + E) time, whereas algorithms like Dijkstra (O((V + E)logV)) and Bellman-Ford (O(VE)) are slower.

The drawback of this, however, is that we need every trust domain router to be aware of every resource in the subtree rooted there. This exacerbates the consistency issues mentioned earlier since the state of the network is duplicated at multiple routers. Additionally, this approach requires messages to be propagated all the way up the hierarchy of routers whenever new clients or switches join, imposing extra overhead.

An alternative approach is to only propagate messages up the tree for links that connect one trust domain to another. With this approach, routers effectively treat the internal structures of descendant trust domains like a black box, and assumes the child routers can find routes between any two member resources (and we assume that trust domains are dense networks with short edges). For each internal node in a trust domain, we collect the GDP names of the resources it links to, as well as the GDP names of the trust domains they belong to. With this information, we can effectively build a reduced version of the network solely consisting of connections between trust domains. We can run a pathfinding algorithm from earlier from the "ends" of the trust domain the joining client is in (Saltzer et al., 1984). After this, we use the path with the shortest distance and send the results back down the tree. Note that trust domain routers still directly maintain connections between clients in their ownership.

This leverages the hierarchy of descendant trust domains that is modified at routers when new trust domains are created. The router for the trust domain of the joining client is responsible for finding a path from the appropriate "end" of the trust domain to the joining client, and it can use a similar algorithm to the lowest common ancestor. This approach breaks the problem of building a multicast tree into cross-domain and intra-domain path finding and does not require full global visibility of all GDP resources unlike the previously mentioned approach (just the trust domain hierarchy rooted at the lowest common ancestor and the "ends" of trust domains, i.e. where they connect to other trust domains). The drawback of this approach is that it can still suffer from consistency issues when resources leave or are added, albeit not as much as those that require global visibility at the lowest common ancestor.

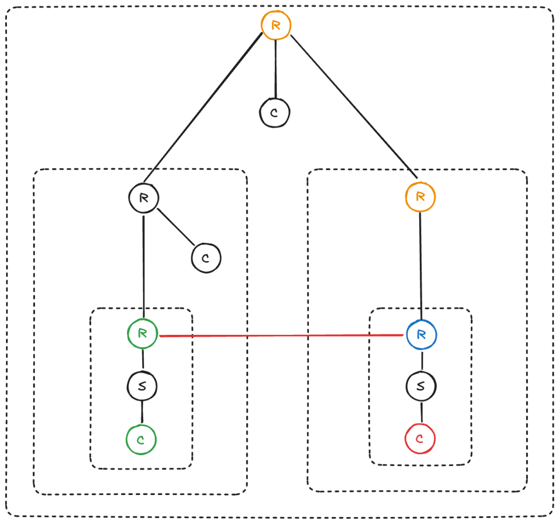

Figure 1: In this diagram, the green router and client start out as part of a multicast tree. The red client wants to join, so it messages the blue router, which in turn propagates messages to the two orange routers. The top orange router then observes that the red edge is the shortest path. The top-level router only sees the other routers and the links between them.

2.5.3 Message Sending

In any case, the actual sending of messages to a multicast group is done via a flood-based algorithm, where each node sends the message to all other connecting nodes that are also part of the tree, aside from the sender. We deliberately did not tailor our design specifically for DataCapsules. While DataCapsules are the main units for data transfer between members of the GDP, we wanted our approach to be generalized, especially since applications like Paranoid Stateful Lambda (Chen et al., 2022) and FogROS (Kaiyuan et al., 2021) gain a major upside by having a multicast tree. Thus, we used a generalized notion of a message to be sent between members of a multicast group.

Message Encryption Upon joining the multicast group, the new node sends a message to the leader

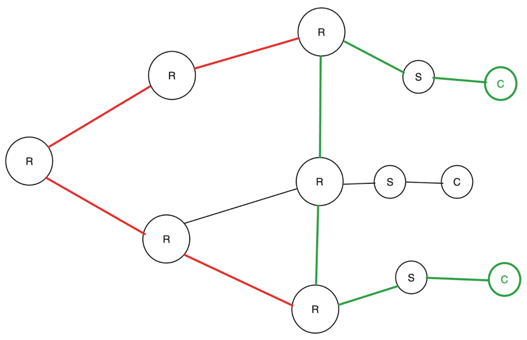

Figure 2: The diagram shows how trust domains without a client involved in the multicast group (shown in green) can be used to find a shorter path between trust domains than going to the root router (shown in red). This practice motivates the need for end-to-end encryption of messages, ensuring that only the intended clients can read them.

client asking for keys, along with its certificate. The leader responds with keys encrypted under the client's public key (under a nondeterministic scheme like RSA with OEAP for IND-CPA security). There are two options to verify a client's authorization. The first requires the trust domain router to ask the certificate authority if the client is allowed to join the multicast group before any pathfinding occurs. However, this requires the certificate authority to be operational at all times.

To ease the requirements on the certificate authority, we can allow pathfinding to happen first and get the joining client to multicast a message asking for keys. If the client is not authorized by the leader to join the multicast group, the client can deny the join request and send a message to the trust domain the client is in. The trust domain router then runs a protocol to make the client leave the multicast group, which can lead to the multicast tree being pruned (this protocol is shown later). The downside of this approach is that it requires action from multiple components of the GDP and is more complex. Additionally, it leads to the possibility of an edge case where one authorized and one unauthorized client attempt to join concurrently. This leads to the need for synchronization between the tree pruning for the unauthorized client and the pathfinding for the authorized client.

Key Rotation and Revocation of Access Key rotation is handled by having the leader client multicast a message under the old key. To handle clients leaving the multicast group, we have clients inform their domain router that they are no longer interested in the multicast group. This causes a request for key rotation to be multicasted to the group. Revocation of access to a client is also supported, and in this case the router for the trust domain directly performs the above procedure. When the keys are rotated, the client will not receive the message containing the new keys, although that message will still be encrypted with an old set of keys.

2.5.4 Preserving Multicast Groups When Resources Leave

Trust Domains When a trust domain leaves the hierarchy, it sends messages up the trust domain hierarchy that it is leaving. In this case, it has two options: disconnect the entire subtree rooted there, or hand off its subtrees to another trust domain that is below its parent. The changes are sent to all direct ancestors of the leaving trust domain and reflected in the adjacency lists they maintain. The children of the leaving trust domain request to rejoin any multicast groups they have information about to ensure that the multicast trees are not broken. Additionally, trust domain routers may have outgoing links that are not part of the tree of trust domains. To ensure all topologies are up to date, we require that the leaving trust domain send a notice to all neighboring trust domains connected with links that are not part of the hierarchy.

Switches When switches leave a trust domain, they are removed from the RIBs of the trust domain they belonged to. If the switch linked to something outside its trust domain, messages are propagated to the trust domain's direct ancestors and the adjacency lists both within and above the trust domain are modified. Additionally, any multicast groups that use that switch (known to the trust domain owning the switch) are rejoined by the trust domain.

Clients When clients leave a multicast group, they send a message to the router of their trust domain. The path to that client is erased from the trust domain's RIB. To avoid sending messages on unnecessary links, the corresponding multicast tree is pruned if there are no clients in the multicast group left in the hierarchy of trust domains rooted there (which can be measured by the number of child trust domains that have information about the multicast group), and there are one or fewer outgoing edges for that multicast group in the trust domain. The trust domain propagates a message up the tree of trust domains saying it no longer has information about the multicast group, and the procedure is repeated at the trust domain receiving the message. A message is also sent to neighboring trust domains that may not be in the multicast tree before the link is ultimately removed from the multicast tree (and repeated at the neighbor if applicable).

Leadership Handoff If the leader client leaves its multicast group, we do a handoff of leadership followed by a key rotation. Handing off leadership involves multicasting a message about the leader's intent to leave, and appointing the new leader to be the first client to respond (and multicasting a message verifying who the new leader is before leaving). Additionally, the existing leader sends a message to the certificate authority endorsing the first client that responded as the new leader. Therefore, each multicast group has a chain of signed messages from leaders of the group. Using this chain of messages, trust domain routers can still honor authorizations from previous leaders of the multicast group.

2.6 Fault Tolerance

2.6.1 Switches and Links Within a Trust Domain

Trust domain routers are responsible for checking functionality of resources in their domains. If a trust domain router detects a failed switch or link, it is responsible for ensuring all multicast trees that use the switch/link have an alternate route. For switches and links that connect within a trust domain, this can be accomplished by placing the switch/link on a "watchlist" and adding it back when it is operational. Resources on the watchlist cannot be used for sending messages and the root router recomputes the routes for the "ends" that each multicast tree uses. Notably, in this approach, trust domains are responsible for checking the health of their resources. For switches and links within a trust domain, a heartbeat can be sent periodically by flooding the network, and a new route can be computed for any multicast groups using failed switches or links.

2.6.2 Links Between Trust Domains

Recall that the topology of trust domains includes a tree of trust domains with possible edges between trust domains that are not part of the tree itself. To test the functionality of non-tree links (not leading to a parent or child trust domain), we send heartbeats along the links and expect a response. If a response is not received in a timely manner, the trust domain sending the heartbeat requests to rejoin any multicast groups using the link.

2.6.3 Trust Domain Routers

If a trust domain router goes down, it can affect the ability of its children to create and join multicast groups or receive messages. Thus, we have two choices for what to do: wait for the router to recover and tolerate some amount of missed messages, or immediately repair any affected multicast trees.

2.6.4 Parent Failure

In the latter option, trust domain routers periodically send a "heartbeat" message to their parent and children. If a router does not receive a heartbeat from its parent, it needs to reattach to another trust domain in order to preserve multicast groups. In this case, the router can send requests to become the child of a neighboring trust domain on non-tree edges. If the neighboring trust domain accepts, the edge becomes a tree edge and the trust domain attaches to its neighbor. Additionally, it requests to rejoin any multicast groups it participates in. This approach is guaranteed not to create a cycle in the hierarchy of trust domains since the edge between a trust domain and its parent is moved. This approach permanently affects the hierarchy of trust domains and can lead to imbalances in the tree of trust domain routers. To deal with this problem, we allow a trust domain router to reorganize the topology rooted there by requiring certain child trust domains to attach to another child trust domain. These changes must also be propagated up the hierarchy of trust domains. Finally, the child whose parent failed rejoins multicast groups it has information about. This repairs any multicast trees that went through the child.

2.6.5 Child Failure

A downside of the above approach is that failed trust domains with healthy parents can miss a lot of messages or fail to send messages for multicast groups they handle. To minimize missed messages, routers can maintain a queue of messages intended for failed children (this can be recorded if a parent does not receive a timely heartbeat response from its child). If a child router reattaches (determined by sending a heartbeat message or a join request from a child marked as needing message storage), the parent router forwards all the messages to its child.

2.6.6 Multicast Group Leadership Handoff

An edge case emerges when a trust domain containing the leader client of a multicast group fails. Since the leader is responsible for key distribution and access control, those capabilities are removed if the trust domain containing the leader fails. To this end, we outline a method to hand off the leadership of a multicast group in the event of a failure. If the trust domain containing a multicast group leader fails, the parent of that trust domain will detect the failure through its heartbeat. The parent will then check if it has any clients in that multicast group. If it does, the parent selects a client to be the new leader and gets that client to conduct a key rotation. If the parent does not have any clients in the multicast group, it propagates the message up to its parent (and so on). If no client owned by a direct ancestor of the failed router is found to be a member of the multicast group, the handoff fails and the multicast group continues to exist without key distribution or access control capabilities.

2.6.7 Dealing With Consistency Issues

One of the most notable drawbacks of having eventual consistency over the topologies stored in trust domain routers is that multicast trees can be built using resources that are about to leave (or have already left and not reflected the changes in their ancestors). To deal with this, if a trust domain router sends a request to join a multicast group and does not receive its encryption and HMAC keys within a particular time frame, it can resend the request a specified number of times before giving up.

2.7 Path Optimization

Some edges may be part of multiple multicast trees because they offer a convenient link between groups of trust domains that wish to communicate. This can lead to edges being overutilized. If an edge is overutilized, its latency may increase, and it is beneficial if some trees stop using that edge. Additionally, new trust domain routers and switches can frequently be added to the GDP, and the multicast trees should adapt to find shorter paths if they exist, while balancing the load of sending messages with existing paths. In this section, we propose a solution to both problems.

To deal with both issues, we allow a router in a multicast tree to voluntarily ask for a new path based on a vote of its interested clients (this can be a simple majority vote over healthy clients that are members of the subtree rooted at a trust domain) and rejoin the multicast tree. Since the router knows the routes for all multicast groups that have clients owned by the subtree of trust domains rooted there, it can send a message to all such clients (which respond with a yes or no vote). While the new route is being computed, the existing route is still used to minimize missed messages as a result of path optimization. If a new path is found, the existing multicast tree is pruned as if the trust domain had lost all clients in the multicast group and the new path is added.

We can use a similar approach to request new paths due to congestion. Since every switch is owned by a trust domain, we let trust domains monitor how many messages per some period of time are sent or received along links corresponding to switches they own. If the number of messages is above a threshold specified by the trust domain, it can request a new path with the same mechanism above. The trust domain router requesting a new path propagates a message up the tree until a trust domain with visibility over both ends of the link is found. This ancestor domain then attempts to find another path between both ends of the link. This approach only allows a single congested edge between trust domains to be bypassed. Additionally, it requires modifications to our flood-based routing approach. Instead of indiscriminately flooding an edge, the load needs to be divided between the old path and any new paths. If additions to the multicast tree use edges on the new path, they need to be converted to receive all traffic for the multicast group.

3 Implementation

We have implemented a simulation of our protocol and related algorithms in Python. This section examines the specifics of the implementation in relation to the design of our multicast protocol. We focused on implementing tree building, message encryption, and tests for stress on routers. For simplicity, we serialize events at each router. Our implementation is available at https://github.com/mikkelsvartveit/gdp-multicast-simulator.

3.1 Establishing Multicast Groups

A multicast group is created by a client sending a message of type CREATE-MULTICAST-GROUP to its domain router. The domain router creates a record for the multicast group in its RIB. Each multicast group has a lowest common ancestor (LCA) router. The LCA is the lowest router in the domain hierarchy of which all members of the multicast group are descendants. The LCA router is responsible for building a spanning tree between all domains that contain members of the multicast group. By utilizing this concept of an LCA router, we can ensure that tree building is kept as far down the router hierarchy as possible, which helps with global scalability. Upon creation of a new multicast group with a single member, the LCA router is, naturally, the domain router of the client that creates the group. The router will also propagate a message up the router hierarchy letting all ancestor routers know about the fact that a multicast group with this GDP name exists, as well as which router is currently the LCA.

3.2 Joining Multicast Groups

If a client wants to join an existing multicast group, it sends a message of type CLIENT-JOIN-MULTICAST-GROUP to its router. The router updates its local multicast group record to include the joining client as a member, and runs the Dijkstra algorithm within the domain to find the shortest path from the joining client to any of the existing members of the group. Additionally, the joining client's router takes an action depending on whether or not its domain is already part of the multicast tree, and where in the hierarchy the LCA router lies.

- If the joining client's domain is already part of the multicast group, the LCA already knows that messages should be sent to the domain. Therefore, no further action is required.
- If the joining client's domain is not already part of the multicast group but its router is in fact the LCA, the router computes a path from itself to the rest of the spanning tree of domains.
- If the joining client's domain is not already part of the multicast group and the LCA router is an ancestor of the joining router, the joining router sends a message of type ROUTER-JOIN-MULTICAST-GROUP to the LCA router. The LCA router then computes the shortest path from the joining client's domain to the rest of the spanning tree of domains, again using the Dijkstra algorithm.
- If the joining client's domain is not already part of the multicast group and the LCA router is a descendant of the joining client's router, this router becomes the new LCA. The joining client's router sends a message to the current LCA of type MULTICAST-GROUP-TRANSFER-LCA, upon

which the current LCA transfers the existing spanning tree of domains to the new LCA. The new LCA now computes the shortest path from itself to the other domains in the tree and adds this path to the spanning tree.

- If the joining client's domain is not already part of the multicast group and the LCA router is neither a descendant nor an ancestor of the joining client's router, a new LCA has to be found. This is done by propagating a message up the domain hierarchy until a router that knows about the multicast group is found. This router becomes the new LCA, and runs the Dijkstra algorithm to find the shortest path between the current spanning tree of domains and the joining client's domain.

3.3 Sending Multicast Messages

To send messages to a multicast group, recall that we make the distinction between intra-domain routing and cross-domain routing. Each router is responsible for facilitating intra-domain routing to the members of the group that fall within their domain, while the LCA router facilitates cross-domain routing for the entire multicast group. When switches receive a multicast message and don't have the routing information in their routing table, they query the RIB of their local router to get the next intra-domain hops. When a multicast message arrives at a router for a group not in their routing table, they query both their own RIB as well as the RIB of the group's LCA router. This allows the routers to correctly forward the message both intra-domain and cross-domain. With each node now equipped with the knowledge of where to forward the message, reaching all clients of the multicast group is as simple as flooding the multicast tree.

4 Evaluation

4.1 Comparison with PSL Multicast Paper

Our work builds upon that of the PSL multicast team (Plutowski et al., 2021), who attempted to build a multicast tree serving nodes of Paranoid Stateful Lambda, a key-value store built on top of the Global Data Plane. Their multicast was also flood-based and assumed a hierarchy of routers, but imposed more strict network topologies between nodes. For example, they did not consider that direct connections between clients in different trust domains could exist, and forced multicast messages to operate only in the tree of router hierarchies. While our design incorporates similar ideas, we offer the possibility of using the full network topology instead of just the connections between routers.

Additionally, our hierarchical structure abstracts certain functions away from ancestor routers. The PSL multicast paper devoted significant attention to creating a balanced tree, which could involve moving clients to a different router (and therefore modifying the hierarchy). By contrast, our design aligns more closely

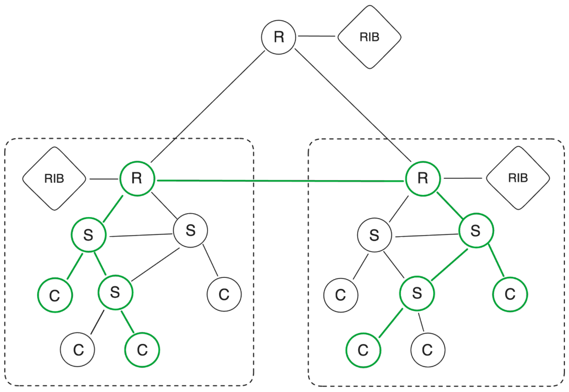

Figure 3: Topology Used to Test RIB Size

with concepts from the Global Data Plane, allowing the owner of a trust domain to coordinate messaging within that trust domain as they see fit. Another benefit of our design is that faults within a trust domain can be handled by the owner of that trust domain instead of involving other routers. Since the Global Data Plane is meant to be adaptable to a number of different use cases within trust domains, we believe our design fits the brief better.

Our approach also differs in terms of security. We include a method for routers to certify the identities of clients attempting to join multicast trees, given a globally visible certificate authority. The capabilities for end-to-end symmetric encryption, rotation of keys, and revocation of access, are new features offered by our proposal. We ensure that only members of the multicast group are authorized to read messages sent to that multicast group, thereby providing tolerance for false positives (where a message is sent to an unintended recipient).

4.2 Multicast Tree Properties

We first verify some desirable properties of our multicast tree design. For this test, we use the topology shown in Figure 3, where green resources and links signify membership of the multicast tree. We verify that parent RIB sizes (measured in bytes) do not scale with the number of clients in the multicast group. Child RIBs, on the other hand, should scale with the number of clients since they need to maintain state for those clients. This illustrates the "end-to-end principle" for trust domains we rely on in our multicast tree protocol, where parent routers effectively treat child domains' internal structures as a black box. Figure 4 shows that our design obeys this principle. We also examine the capabilities of our multicast tree to leverage non-tree edges. We calculate the sums of edge costs for each message sent, where an edge between trust domains has weight 100 and an edge within a trust domain has weight 1.

We compare our design against an approach from the 2021 PSL multicast paper, namely, one that only uses edges in the tree of trust domains for routing. We use

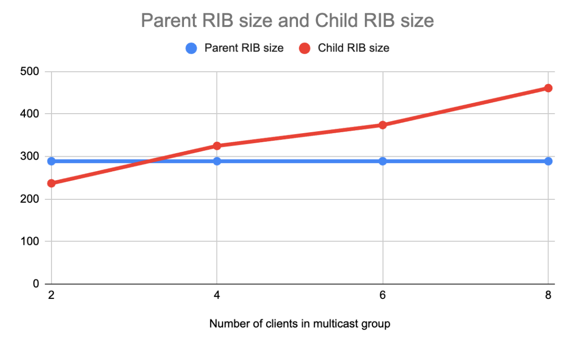

Figure 4: RIB Sizes for Varying Numbers of Clients

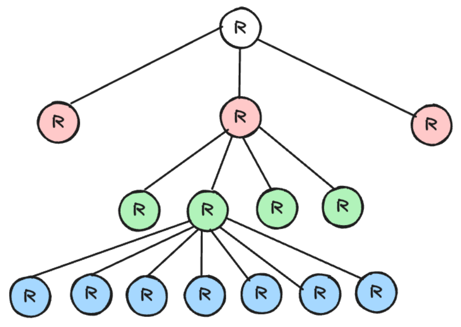

Figure 5: Topology of 100 Trust Domains and 1000 Clients

the topology in Figure 5, with 1 router connecting to 3 children, which each connect to 4 children, which each connect to 7 children. At each router, we add 5 switches and 10 clients, for a grand total of 100 trust domains and 1000 clients. When considering non-tree edges, we added an edge between trust domains on the same level of the tree with probability 0.25.

We ran 1000 trials, where each trial consisted of randomly choosing 10 clients to participate in a multicast group and sending a particular amount of messages to that multicast group. Notably, for this test, the non-tree links are different for every trial. The data show that our multicast tree building protocol offers a significant reduction in total edge weight relative to a protocol that does not consider non-tree edges. We initially see small gains from small amounts of messages, but are able to achieve roughly a 34% reduction (see Table 1) in mean edge weight needed when 1000 messages are sent to the same multicast group. Since we used a randomized topology for each trial, results in practice could be better or worse depending on how well non-tree edges align with groups of clients wanting to join a multicast group.

Additionally, we find that as more messages are sent in our topology, the difference between our design and a hierarchical approach tends to widen.

4.3 Security Overhead

Our design places an emphasis on end-to-end symmetric encryption, but this comes at the cost of key distribution. We use the same topology as the previous test, but without the non-tree links. We conduct 1000 trials of creating a multicast group from 10 randomly selected clients and sending a certain number of messages in that group. In keeping with previous tests, we measure the total edge weight from each trial and present the mean.

As the number of messages increases, the relative overhead of key distribution decreases (see the quotients in 2). This makes sense because key distribution is conducted when new clients join – hence it scales with the number of clients in the multicast group. However, this test shows that if the number of messages sent is significantly greater than the number of clients in the multicast group, encryption does not create much additional overhead.

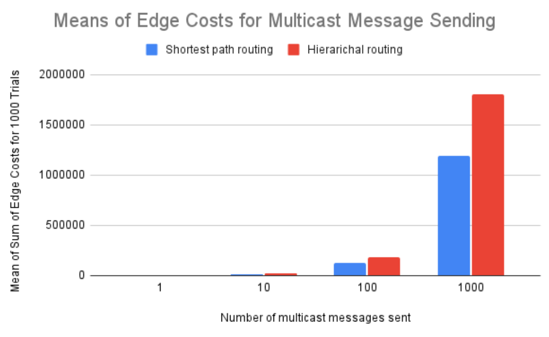

Figure 6: Mean of Edge Costs for 1000 Runs

Messages	Our Design	Hierarchical	Quotient
1	7696.22	8906.43	0.864
10	18334.57	25098.35	0.731
100	125286.92	187017.53	0.669
1000	1193208.33	1806209.33	0.661

Table 1: Means of total edge cost from 1000 runs of our design and a strict tree hierarchy, and the quotient of those two values. Messages refers to the number of messages sent per run.

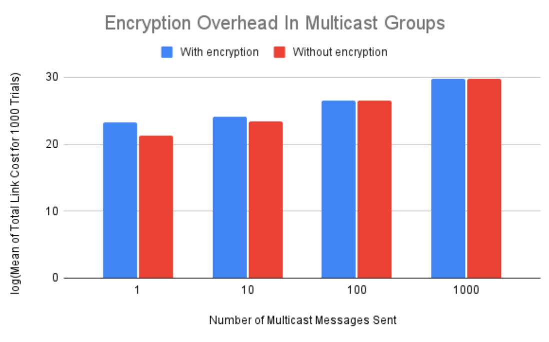

Figure 7: Mean of Edge Costs for 1000 Runs for Encryption and Message Sending

Messages	Encrypted	Unencrypted	Quotient
1	9526967.72	2446360.039	3.894
10	17641352.68	10560745	1.670
100	98785202.29	91704594.61	1.077
1000	910223698.4	903143090.7	1.008

Table 2: Means of total edge cost from 1000 runs with and without encryption, and the quotient of those two values. Messages refers to the number of messages sent per run.

4.4 Stress at Higher Levels

A key feature of our design is that it lowers the stress imposed on routers high in the hierarchy of trust domains. We ran 1000 trials where we made 10 groups with 10 randomly chosen clients each, and sent 10 messages to each group. To quantify stress on higher levels of the topology, we counted the number of messages sent through the root, using the same topology as the previous test.

The mean number of messages sent through the root router using strict hierarchical routing was 213.845, and was reduced to a mean of 22.534 in our design. Interestingly, the standard deviations of messages sent through the root were nearly identical (11.699 for hierarchical and 11.775 for our design). This could be due to the probabilistic nature of our experiment where some clients were located far from non-tree links and thus had to go route through the root.

4.5 Parent Fault Tolerance Overhead

The procedure for fault tolerance when a parent trust domain fails involves messaging along non-tree links to attach to a non-tree neighbor as a child, followed by rejoining multicast groups. Due to time constraints, we could not implement and test this capability directly, but we have already addressed the procedure for joining multicast groups.

5 Future Work

5.1 Comparison of Path-Finding Algorithms Between Trust Domains

Our implementation uses a modified version of Dijkstra's algoritm to construct a path between trust domains (where we specified different topologies and edge weights). We would be interested in seeing the impacts of different path finding algorithms such as DFS to analyze the tradeoff between optimality of the path and efficiency in finding the path when many trust domains and clients are involved (and especially when trust domains or other GDP resources can leave at any time). The case for using an efficient but non-optimal path finding algorithm is that a connection to the rest of the multicast tree can be produced sooner, which is a benefit if resources leave frequently.

5.2 Scheduling at GDP Switches to Support Performance-Critical Applications

Our implementation has the potential to let multiple multicast trees use the same switches to transmit messages. This approach can have deficiencies if the multicast tree is used for a performance-critical application. For example, a smart factory could send messages to all components to shut down in the event of a critical hardware defect. The shut-down message should take precedence over others. A deficiency of our message sending approach is that there is no notion of priority for different multicast groups. There are two main hurdles to solving this problem: ownership and scheduling. For ownership, all members of the multicast tree would need to agree on the priorities of different multicast groups. For scheduling, we can implement priority-based scheduling algorithms at each switch.

5.3 Better Methods of Repairing the Hierarchy of Trust Domains

Updates to the adjacency lists stored in routers are eventually consistent since they need to go through switches and possibly other routers to finally affect an ancestor. If trust domain routers fail, we get each child trust domain of the failed trust domain router to reestablish connections to any multicast groups it was part of. While this approach gets the multicast groups back in working order quickly, it makes many edits to eventually consistent state, potentially leading to more work downstream to repair multicast trees. An alternative approach is to allow trust domain routers to provision new routers with the same state (end-to-end connections, clients, child routers) as a failed child router. This could enable a much simpler method for fault tolerance, but would require parent routers to maintain some visibility over the internals of its child routers.

5.4 More Fine-Grained Synchronization Over Router State

Currently, our implementation serializes actions for each router of a trust domain. However, there is room for improvement regarding the serialization of updates to resources held at trust domain routers. For example, updates that affect sibling trust domains in the hierarchy should be allowed to run concurrently. A possible solution is to use a locking scheme similar to Two-Phase Locking. Each node in the adjacency list could have a lock that operations updating the adjacency list need to acquire. Additionally, locks can be differentiated into read and write locks, where a transaction having a write lock on a particular resource blocks all operations on that resource. We can also enforce a rule that no transaction can acquire a new lock after releasing locks, to ensure that all transactions are serializable. Thread pools can be used to avoid excessive overhead from too many threads at trust domain routers.

References

[Chen et al.2022] Kaiyuan Chen, Alexander Thomas, Hanming Lu, William Mullen, Jeffery Ichnowski, Rahul Arya, Nivedha Krishnakumar, Ryan Teoh, Willis Wang, Anthony Joseph, and John Kubiatowicz. 2022. Scl: A secure concurrency layer for paranoid stateful lambdas.

[Inc.2001] Cisco Systems Inc. 2001. Ip multicast technology overview, Oct.

[Kaiyuan et al.2021] Kaiyuan, Chen, Yafei Liang, Nikhil Jha, Jeffrey Ichnowski, Michael Danielczuk, Joseph Gonzalez, John Kubiatowicz, and Ken Goldberg. 2021. Fogros: An adaptive framework for automating fog robotics deployment.

[Mor et al.2019] Nitesh Mor, Richard Pratt, Eric Allman, Kenneth Lutz, and John Kubiatowicz. 2019. Global data plane: A federated vision for secure data in edge computing. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1652–1663.

[Plutowski et al.2021] Marcus Plutowski, Willis Wang, and Vivek Bharadwaj. 2021. Delegated, sharded, multicast tree for paranoid stateful lambdas.

[Saltzer et al.1984] J. H. Saltzer, D. P. Reed, and D. D. Clark. 1984. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, nov.