Hierarchical Routing on the Global Data Plane

Mira Sharma
Nia Nasiri
Eric Nguyen
Tianyun Yuan
UC Berkeley

Abstract

We present a system of hierarchical Routing Information Bases (RIBs) for the GDP, optimizing for reduced advertisement traffic, scaled numbers of DataCapsule servers, and increased DataCapsule availability as servers go offline and later reappear online. The scalability of GDP hinges on the scalability of its routing system. Previous work [5] implements secure delegated routing with a single global RIB, but routing with a hierarchy of RIBs is better suited for massive scale and better models of real-world nested trust domains. We address the challenges that arise due to the data-centric, location-agnostic guarantees of the GDP, as well as its flat namespace and security focus.

Keywords: Information-centric networking, Global Data Plane

1 Introduction

The evolution of network architectures and the proliferation of data-centric applications have led to the emergence of the Global Data Plane (GDP), an innovative infrastructure aimed at revolutionizing how data is stored, accessed, and routed in a distributed network environment. The GDP is distinct from traditional network models, most notably the Internet, due to its focus on data-centric, location-agnostic communication and a flat namespace, which facilitates seamless data mobility and security.

In recent years, with the exponential growth of data generated by applications ranging from Internet of Things (IoT) devices to large-scale cloud services, the need for an efficient, scalable, and secure routing mechanism within the GDP has become increasingly critical. Traditional routing systems, typically designed for location-based, hierarchical IP networks, are not well-suited for the unique challenges presented by the GDP. These challenges include routing in a flat namespace, ensuring data availability in a dynamic environment with mobile data sources, and maintaining security and trust across diverse administrative domains.

Our work presents a novel approach to routing within the GDP framework, addressing these challenges head-on. By introducing a system of hierarchical Routing Information Bases (RIBs), we propose a solution that significantly enhances the scalability and efficiency of data routing in the GDP. Our approach leverages the notion of trust domains, providing a structure that naturally aligns with real-world administrative and organizational boundaries. This hierarchical structure allows for more effective management of routing information, reducing the overhead associated with route advertisement and discovery, especially in a network with a large number of DataCapsule servers.

A key contribution of our work is the development of source-based routing protocols and innovative routing data structures that optimize for reduced advertisement traffic. This is critical in a network where the location of data can frequently change and the number of data items (DataCapsules) can be vast. We also address the challenges of routing failure and data mobility by introducing robust failure protocols and meta-DataCapsules. These mechanisms ensure that the network can dynamically adapt to changes, maintain data availability, and efficiently update routing paths when servers go offline or reappear.

The rest of this paper is organized as follows: Section 2 provides background on the GDP and related works, laying the foundation for our contributions. In Section 3, we detail the design of our hierarchical RIBs, including the routing data structures and algorithms. Section 4 describes our implementation and simulation setup. We evaluate our system in Section 5, discussing the performance benefits and potential areas for improvement. Finally, we conclude in Sections 6 and 7 with an outline of future work, and a summary of our contributions.

2 Background and Related Works

2.1 Global Data Plane

The Global Data Plane (GDP) [5] is an infrastructure that provides storage and routing capabilities to support DataCapsules. The GDP is a distributed system of storage nodes (DataCapsule servers) and routing nodes (GDP routers) that enables location-independent communication between clients and DataCapsules based on a DataCapsule's unique name.

The GDP network is organized as a graph of Trust Domains (TDs) which are administrative domains like organizations that operate parts of the GDP infrastructure. TDs enable communication restrictions to trusted portions of the network as dictated by DataCapsule owners. The GDP includes a routing fabric made up of GDP routers that enable communication between various entities using flat addresses, rather than IP's host-centric communication. This makes the GDP an information-centric network. Some key differences of the GDP from IP routing:

- Routes and looks up flat, location-independent names rather than IP addresses, which are tied to specific hosts or locations. This allows mobility of DataCapsules, as well as routing to the closest of several replicas of a DataCapsule.
- Uses strict cryptographic delegations and certificates (e.g. AdCerts, RtCerts) to control routing permissions rather than open routing tables. This provides built-in security.
- Maintains routing state in separate, distributed database services called Routing Information Bases (RIBs) rather than individual routers. This allows independent verification of routes.
- Organizes trust domains and routing domains hierarchically to enable policy control and scalability in routing

Currently, the GDP has secure, delegated routing with only one RIB. Our contribution provides such routing with a hierarchy of RIBs, adapting to scale and nested trust domains. We also add optimizations that minimize network traffic.

2.2 DataCapsules

DataCapsules are a standardization for storing data, similar to shipping containers. DataCapsules' design separates security (authentication, confidentiality, and integrity) from trust in service providers and other platforms, allowing the Global Data Plane to harness the computing and storage of a much wider array of machines.

Each DataCapsule has a unique 256-bit name, and immutable metadata that includes a public signature key belonging to the DataCapsule's owner. DataCapsules can only be written to by their owner, and each DataCapsule has only a single owner. For a given DataCapsule, the owner can generate an AdCert, a signed certificate that allows a specific DataCapsule server to host the DataCapsule. When a DataCapsule-server connects to a GDP router, it advertises its own name, and the AdCerts of all the DataCapsules the server is delegated to serve.

Our contribution introduces meta-DataCapsules (see section 3.2), a DataCapsule that each server has, containing all of the server's hosted DataCapsules. This allows for a variety of optimizations that minimize traffic.

2.3 NDN Routing

Named data networking (NDN) [1] is a proposed data-centric Internet architecture, where packets carry a unique name identifying the data they contain. NDN routing protocols propagate the reachability of data names, similar to IP routing protocols propagating the reachability of IP addresses. Unlike IP, NDN routing protocols are themselves NDN applications, with routing updates being named and secured NDN data packets. This provides built-in security.

Each request is carried in an Interest packet. The forwarding module of a given NDN node contains a Pending Interest Table (PIT), a Forwarding Information Base (FIB), and a Content Store (CS). NDN has a stateful forwarding plane because of the PIT, which records interests and upcoming data. This state allows NDN to do routing and forwarding directly based on application data names, without separation between name resolution and packet forwarding.

The CS is the first place the node checks for a Data upon receiving an Interest.

While FIB entries in IP routing only have one next hop, NDN FIBs can have multiple next hops, supporting multipath forwarding. This is possible because the PIT checks the nonce in the packet of each NDN request to detect and stop any possible looping.

The stateful forwarding plane in NDN changes the requirements and importance of routing protocols compared to IP. The FIB is only one input, not the sole input, for forwarding decisions. In NDN, the strategy layer sits between the FIB and actual packet forwarding. This allows the forwarding strategy to be based on more than just routing protocol updates, providing more flexibility.

We compare our contribution with NDN:

- Similar to an NDN routing protocol, our contribution implements routing for a data-centric network with a flat namespace that lacks prefixes and other intuitive ways to build a hierarchy.
- Our design adds a cache, updated as requests flow through the network, that serves as the first place to check for routing information, similar to NDN nodes' Content Stores.
- NDN has native security; our contribution has some native security provided by DataCapsule design.
- Unlike NDN, our contribution does not have a stateful forwarding plane.
- Scalability is also a concern NDN routing attempts to address. Small NDN networks may not need a routing protocol, instead self-learning their data availability.
- NDN routing is for general, all-purpose applications, and names can be user-constructed, with

meaning (not cryptographically secure). On small local networks, NDN names need not be globally unique. Routing on the GDP is for requests for DataCapsules, which have standardized, random, globally unique names.

2.4 FOGROS2-SGC

FogROS2 [2] is an extension of the Robot Operating System (ROS2), a software platform for robotics applications, that provides robots easy access to cloud (and fog) computing.

FogROS2-SGC (Secure Global Connectivity) builds on Fog-ROS2 to allow robots to connect across physical space and untrusted infrastructure. FogROS2-SGC extends GDP, using its location-independent routing for globally unique identifiers in a flat namespace, but for identifying and routing between robots instead of DataCapsules.

In FogROS2-SGC, each router has its own RIB. We decouple this in our implementation; our overlay router and switches are stateless. Since FogROS2-SGC builds on top of GDP's location-independent routing, and our contribution is a hierarchical implementation of that routing, it could potentially be compatible with FogROS2-SGC, and help with scaling.

2.5 DNS

Domain Name System (DNS) is a hierarchical naming system and a vital part of Internet infrastructure. DNS lookups are used to translate internet domain names to IP addresses, which have prefixes with meaning.

Unlike DNS, GDP has a flat namespace. However, we implement a multi-level lookup table for addresses similar to DNS. We abstract away the addresses from the user, so the data-centric, location-independent nature of GDP routing is preserved.

2.6 SDN Controllers

Software-Defined Networking (SDN) [3] separates the forwarding process, on the data plane, from routing, on the control plane. The control plane is made up of SDN controllers, which can follow a hierarchical design. The data plane is made up of network devices such as routers.

Our contribution uses a two-layer routing and forwarding (switching) framework, similar to SDN. Similar to GDP's original routing scheme, SDN's first control plane design involves a single controller, which produced scalability problems for SDN, and led to different multi-controller schemes being suggested.

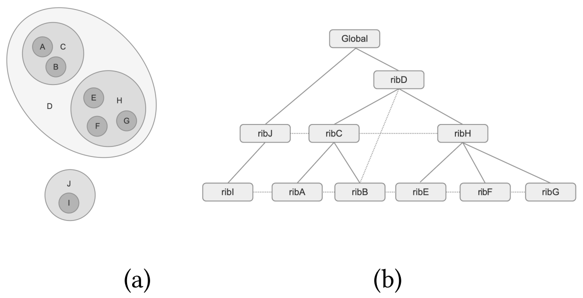

Figure 1. The example trust domain configuration in (a) is translated into a hierarchy of RIBs, with one RIB corresponding to one TD. Solid lines in (b) represent parent-child relationships, while dotted lines represent peering RIBs. Peering RIBs are directly connected through overlay switches, but have no nested TD relationship.

3 Design

In this section, we detail the data structures of the RIBs, our routing and failure protocols, and modifications made to DataCapsule servers for interacting with hierarchical RIBs. We also introduce a per-server meta-DataCapule for updating RIBs and detecting offline servers.

3.1 Routing

We construct our hierarchy based on nested trust domains and use the definition of peering in [4] to create gossip links between peering TDs that do not have a parent-child relationship. Parent-child relationships are reserved exclusively for nested TDs, where the encapsulating TD is the parent of the TD that is encapsulated. Figure 1 shows a simple translation of a TD configuration to a network topology.

Our routing scheme employs source-based routing, as in [4], to reduce the statefulness of switches. We also abstract away inter-domain RIBs by treating each TD as having a single RIB for simplicity - in practice, this is clearly unrealistic, but simple extensions to the presented algorithms can adjust for multiple RIBs within a single trust domain. In source-based routing, a client that wants to reach DataCapsule *D* sends a routing request to an RIB within its TD. The RIB returns the path of trust domains a packet must be routed through in order to reach the DataCapsule *D*, which the client can then embed in all packets sent to *D*. See Figure 2. Network switches can then simply look at the packet header to learn which TD to forward the packet to next. Switches employ protocols like OSPF to efficiently forward packets within trust domains.

Note that dealing with routing through trusted TDs is out of the scope of this paper. We assume all TDs to be trusted in our protocols and leave integration with

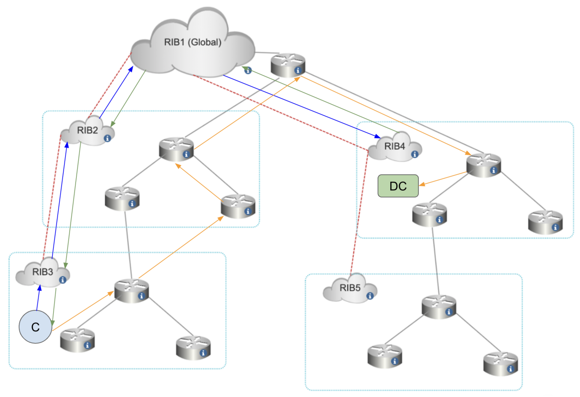

Figure 2. An overview of the source-based routing protocol. Client C wants to reach DataCapsule DC, which is within RIB4's TD. Note that the DataCapsule server that hosts DC is not pictured. The blue arrows represent the path of client C's initial routing request, which returns the routing path of TD 3 –> TD 2 –> GLOBAL –> TD4. The green arrows depict the routing path being sent back to the client, and the orange represents a packet that the client is then able to send to DC.

trusted routing algorithms to future work. We also assume that Trust Domains are static and do not consider the case in which RIBs must be moved.

3.1.1 Routing Data Structures. Each RIB in our hierarchical scheme contains two key data structures: the ServerStore and the DCStore. Both are hash tables with cache-like functionalities, with the ServerStore hashed on the server name and the DCStore hashed on the DataCapsule name. The hash keys in the DCStore are not unique, as a single DataCapsule name may correspond to multiple replicas with distinct routing paths. Servers, however, are unique. A single ServerStore entry contains information about that server's mDC checksum (see section 3.2), a list of all DataCapsules hosted by that server that are also within the DCStore, and valid and LRU bits. A single DCStore entry contains information about the DataCapsule's next hop and server, as well as a TD path constructed through advertisement and valid and LRU bits.

The TD path is similar to the notion of an AS path in BGP, where each trust domain that a packet must pass through to get from DataCapsule to a RIB is treated like an autonomous system. Maintaining this TD path prevents routing loops: if a RIB receives an advertisement for an entry that already contains that RIB's name in it, then it discards the entry. A RIB adds its name to the the end of the TD path before forwarding an advertisement to its parent RIB. We also use the length of this path as a metric for route selection, as this number is an upper bound on the number of trust domains we must route through in order to reach the destination DataCapsule.

The next hop field is also updated during the advertisement. When a DataCapsule is advertised to an RIB, the RIB can see the TDs that the advertisement has already passed through, including the DataCapsule's home TD. The RIB can then select the RIB within the trust domain closest to the home TD that it shares a connection with. RIBs also periodically gossip with other peering RIBs, disseminating updates to its DCStore and ServerStore across the network.

3.1.2 Routing Algorithm. To best illustrate our routing protocol, we use an example where a client in TD A intends to route data to DataCapsule 1 (dc1) in TD G. The client sends a routing request to TD A's RIB (ribA), which begins the recursive construction of the routing path.

ribA searches its DCStore for dc1, which may yield multiple matches. To refine the selection, we filter these matches to retain only those instances where dc1's valid bit is set to 1. Should no valid entries emerge from this filter, we escalate the routing request to ribA's parent. If we are currently in the global RIB, there is no parent to forward the request to and we thus return a failure message back to the client.

With the remaining valid entries, we retrieve the associated server names in the RIB's ServerStore and once again filter out DataCapsules with invalid servers. This step is where the value of the ServerStore lies - when a server becomes invalid, either through manual updating or missed heartbeat messages, only a single bit has to be flipped in the RIB in order to prevent further routing to the server. The alternative is to invalidate each DataCapsule in the DCStore hosted by that server, which would require a linear scan over the entire DCStore since it is hashed by the DataCapsule name. This could lead to us sending an incorrect routing path back to the client, where the error would only be caught when a packet attempts to reach the offline server, requiring another round-trip of a routing request and reply.

If there are any remaining DCStore entries, we choose the one with the shortest TD path as the optimal path, add the current RIB's name to the routing path, and forward the request with the updated routing path to best match the entry's stored next hop. The next hop, another RIB, will then repeat the process. Once the next hop is the same as the current RIB, we know that the DataCapsule is hosted by a server within the RIB's trust domain, and at this point, we can send the routing path back to the client.

If no route can be found at any point in the algorithm, a failure message is returned to the client. The client may resend a routing request to retry. Figure 3 shows a detailed example of how requests flow through the hierarchy.

3.1.3 Routing Failure.

DataCapsules may be mobile and the information in RIBs may be temporarily stale, so it is likely that a packet fails somewhere along the routing path. Once again, we turn to an example to illustrate failure protocols.

Suppose a routing failure transpires at a switch located within TD C while routing towards a DataCapsule in TD F, as a result of the next hop's sudden invalidity. In this instance, the switch queues the packet for re-routing and enters the incorrect routing path embedded in the packet's headers in a failure hash. Let's say that this path is [TD A → TD C → TD H → TD F]. The hash's value is initially null to indicate that re-routing is currently in progress. If this path already exists in the hash, it indicates a prior failure along the same route. In this case, we will wait for the re-routing to complete or use the updated value for the hash key as the corrected routing path. A timestamp accompanies each entry in the hash, allowing for its eventual removal after a specified time interval.

Re-routing is a simple matter of querying the current TD's associated RIB for a routing path to the intended destination. The RIB handles this as any other routing request and sends the new route back to the server. If this route is the same as the previously failing one, the RIBs have yet to be updated about the failure state in the next hop. The switch periodically resends the routing request until a new path is obtained. In cases where no route can be established or the resends time out, the switch sends a failure message back to the client.

Assuming the discovery of a new path—say, [TD C → TD D → TD B → TD E → TD F]—this new path is appended to the existing valid path, resulting in [TD A → TD C → TD D → TD B → TD E → TD F]. The newly formed full path is then added to a failure correction hash, where the key signifies the new path and the value represents the previous one. This mechanism ensures that subsequent packets bearing the key's routing path have successfully adopted the corrected route. Similar to the failure hash, each entry in the failure correction hash is accompanied by a timestamp. Upon the expiration of a predetermined time interval, these entries, along with their corresponding entries in the failure hash, are systematically removed.

Subsequently, the re-routing queue is monitored for packets bearing the old routing path, which are then rerouted with the new path. Employing the reverse of the initial valid path, we transmit the newly established routing path back to the client. Assuming successful receipt and subsequent adoption of the new path by the user, we reference the key's value to remove the old path from the failure hash.

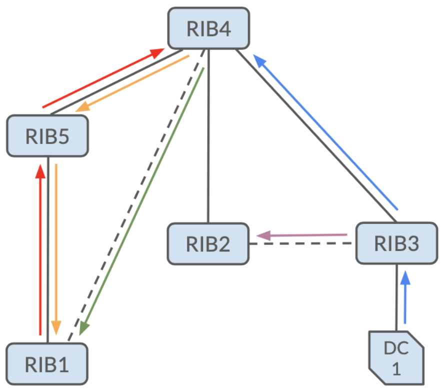

Figure 3. ● A replica of DC1 is stored under RIB3. A request is sent to update RIB3, which also forwards up to the global RIB 4. An ACK is sent backward along the same route. ● RIB3 gossips with RIB2, so if RIB2 GETs DC1 later, it can directly go to RIB3 instead of going up to RIB4. ● RIB1 GETs DC1, defaulting to querying its parent, RIB5, which queries the global RIB4. ● RIB4 sends a request (containing best route from RIB1 to DC1) to RIB1. ● RIB4 sends a request to update caches backward along GET's route. RIB1's cache is updated with RIB4 as the next hop for DC1. If RIB5 had a direct connection to RIB3, its cache would be updated with its next hop as RIB3.

3.1.4 RIB Cleanup.

Our routing scheme ensures efficient control of the routing table's size by leveraging valid bits. Specifically, we employ a mechanism inspired by an LRU (Least Recently Used) cache, which prioritizes the removal of invalid entries over valid ones. This process operates in the background. When an entry is removed from the Server Store, we access the entry's associated DataCapsule list and eliminate each listed DataCapsule from the DataCapsule Store.

3.2 Meta-DataCapsule

In order to easily detect changes in a server's set of hosted DataCapsules and to minimize advertising traffic, each server has a meta-DataCapsule (mDC) that contains all of the server's hosted DataCapsules in a hash table hashed on the owner's public key. Additions, deletions, and modifications of DataCapsules in the server are received and verified by the server and then pushed to the mDC. The server's corresponding RIB is notified of changes through periodic heartbeat messages, which

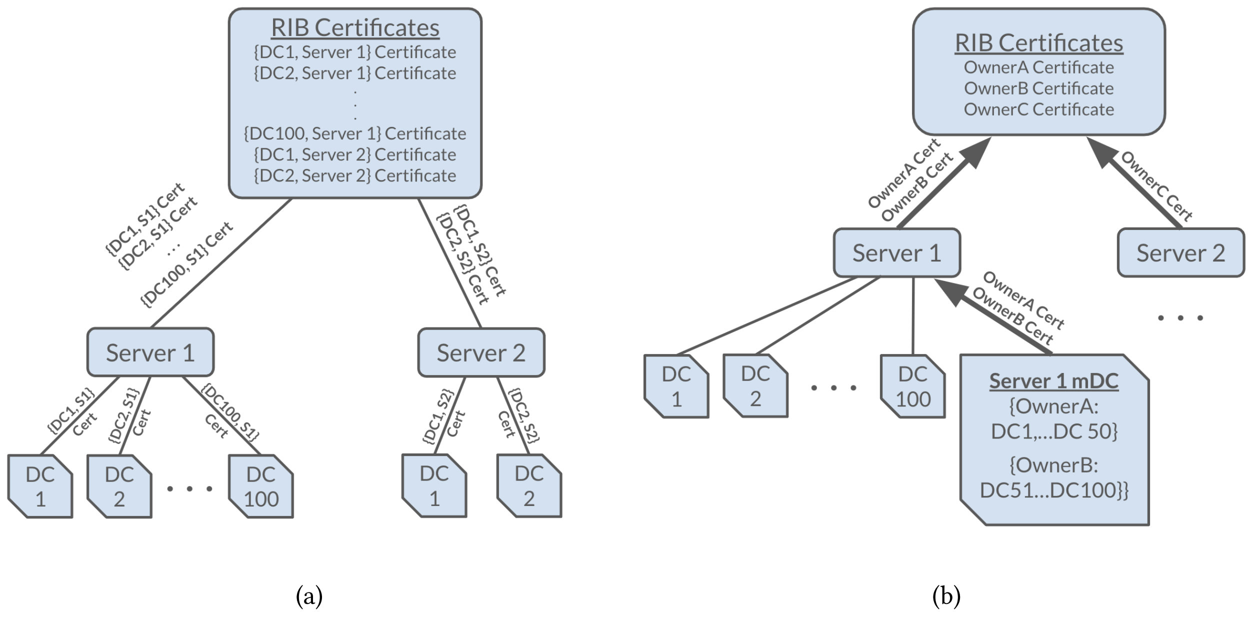

Figure 4. Figure (a) shows certificate-related traffic in the current server implementation, while (b) shows how aggregation by the DataCapsule owner reduces the number of messages that need to be sent. This is particularly important when we need to re-validate the entire server, such as after a server crash, as this traffic flows upwards through the hierarchy, and certificate validation takes a nontrivial amount of time.

contain a checksum of the server's mDC and are verified against the last checksum for that server in the RIB's Server Store. If the heartbeat checksum does not match the RIB's stored checksum, the RIB can query for updates. Changes to the server's DataCapsule set that have not already been propagated to the RIB are stored in a server-side buffer, which is serialized and sent to the RIB upon query and is then cleared. Using checksums and heartbeats instead of sending updates as they happen allows for RIBs to quickly detect when servers are offline, anticipate future updates, and minimize advertising traffic.

It is much easier to aggregate certificates within the mDC than across the entire server, allowing for several additional optimizations to advertising traffic. Because we hash on the owner's public key, it is simple for us to aggregate certificates for a single owner within the trust domain, allowing us to minimize the number of certificates that must be sent to and verified by the RIB as shown in Figure 4. This is particularly beneficial when large numbers of DataCapsules are owned by the same entity, for this turns the certification of n DataCapsules into a single verification by the network.

We also reduce advertising network traffic and verification latency by batching multiple AdCerts into a single RIB-bound request, which allows the RIB to verify the certificates in parallel. We are able to do this batching because of the server-side buffer, which enables us to see new DataCapsules or certificates requiring verification and send them in a single message to the RIB.

4 Implementation

We implemented the routing algorithm, including custom RIB, switch, and DataCapsule structures, in roughly 1100 lines of C++ code. We then simulated a multi-hierarchical network as a proof of concept. We use the ZeroMQ library for packet transmission and simulate latency through the "sleep" syscall.

4.1 Network Hierarchy

Through a Python script, we generate network hierarchies into configuration files for a hierarchical network simulation, using the NetworkX library. The network consists of routers, switches, and clients organized in a hierarchical structure. The script establishes peer relationships and latency values for routers, assigns addresses to network entities, and generates configuration files. It models the interconnections between routers, switches, and clients, creating a structured environment for testing and experimentation in network simulation. The resulting configurations capture the simulated network topology, enabling further analysis and evaluation.

There are some downsides to using a simulation-based approach for evaluation, such as difficulties in modeling realistic network latencies and traffic congestion. The latencies presented in the evaluation are meant to explain relative overheads incurred by various routing components and do not necessarily reflect real-world times. However, they provide valuable insight into the tradeoffs made by our approach, and a real-world implementation remains as future work.

4.2 Simulation

We initialize the components of the network through another Python script, which parses the generated hierarchy and runs corresponding C++ binaries. The C++ processes talk to each other through the ZMQ interface. We mark the time difference between routing requests and responses, in milliseconds, for further evaluation.

5 Evaluation

We evaluate our hierarchical routing implementation on several network topologies with different hierarchical levels. As stated in Section 4.3, our evaluation will be based on a simulated network. Each GDP router and RIB are independent processes that communicate with one another through the ZMQ interface. The benchmarks are ran on a Microsoft Azure VM with a 2-core 4-thread Xeon 8272CL 2.6GHz processor. Our main metrics are determining how much overhead multi-level RIBs incur based on latency and how multi-level RIBs reduce the number of requests sent to the global RIB.

5.1 Multi-level Hierarchy Overhead

Hierarchical routing will obviously incur higher latency than flat routing. We evaluate how much overhead is incurred when we introduce multi-level RIBs. Figure 5 shows a breakdown of how much overhead is incurred within the RIBs (control plane) and the GDP routers (data plane). For our test, the network is built as follows. A n-level hierarchy represents a full binary tree with height n. We measure the latency of an average GET request between two random clients across several networks structured as an increasing n-level hierarchy for $n = 1, 2, 3$. From Figure 5, we see that there is a linear increase in latency as we increase the height of the tree of RIBs in both the times for finding the route to a DataCapsule and the time for forwarding a packet to the DataCapsule. This indicates that the overhead incurred by introducing multi-level RIBs is linear with respect to the number of levels and is not too severe.

5.2 Mini-Benchmark

Our mini-benchmark workload consists of x randomly-generated PUT requests and $2x$ GET requests. Each PUT request puts a random DataCapsule name into a local DataCapsule-server. The sender of each PUT request is randomly assigned to a client who connects to its nearest GDP router. Like in the earlier subsection testing average GET request latency, the types of network structures we will test our benchmark on are n-level hierarchical RIBs, which is essentially a full binary tree of height n where each node represents a RIB. Furthermore, each RIB represents a Trust Domain or subdomain (a Trust Domain within a Trust Domain). Within each Trust Domain, there are 4 GDP routers. For intradomain routing, the GDP routers use a learning protocol similar to OSPF. During the router's start-up, it exchanges link state advertisements (LSAs) with neighbor routers in order to build a complete topology of the routers within its local Trust Domain. In our evaluation of the mini-benchmark, we do not include the router initialization/start-up time.

After all x PUT requests have been assigned to random clients, we let each client send the request(s) to their nearest GDP router. Once the DataCapsule name has been advertised and propagated to the local RIB, parent RIBs, and any of its peering RIBs, we kick off the second phase of our mini-benchmark. Each of the $2x$ GET requests randomly requests one of the x DataCapsule names that have been put into DataCapsule-servers distributed across the network during the first phase. Again, the sender of each GET request is assigned to a random client in any Trust Domain. Once each GET request is assigned to a client, we let all clients send their queued GET requests all at once to their closest GDP router. We measure the time it takes from this point to the point where all GET requests have been responded to and reach back to the sending clients. We evaluate

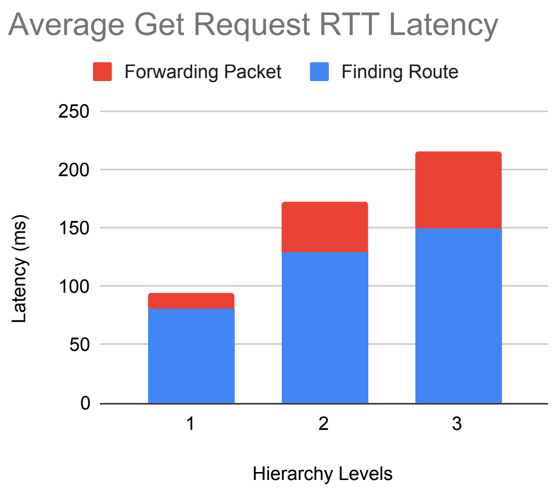

Figure 5. Average Message RTT latency for increasing hierarchical levels including both time to fetch the route from RIBs and forwarding packets through GDP routers

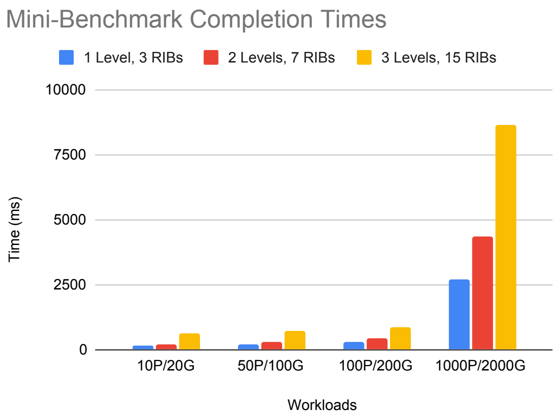

Figure 6. Benchmark completion times for multiple network topologies with increasing number of multi-level RIBs

n-level hierarchical networks of $n = 1, 2, 3$ with this benchmark.

Figure 6 shows the completion times for each additional level added to the hierarchy. The x-axis labels are the workload formatted as $\{x\}P2\{x\}G$, indicating x PUT requests and $2x$ GET requests. We did not expect the 3-level hierarchy to have such high completion times for the 1000P/2000G workload. We hypothesize the cost of process context-switching can be attributed to its completion time. In the future, we expect to test our implementation with actual routers or multiple machines to eliminate the overhead of process context-switching when testing on a single machine.

Figure 7 highlights a key benefit of multi-level RIBs, taking most of the load of GET requests going to the global RIB and handling them at local RIBs. For this metric, we re-run the benchmark and measure the number of route requests going to the global RIB and test our hierarchical implementation against flat routing. In the hierarchical case, the global RIB is the root node in our n-level tree of RIBs. During this run, we use destination-based routing for both the hierarchical case and flat routing case to ensure a fairer evaluation. Initially, all routers have empty FIBs, requiring a request to its local RIB to figure out the routing path and to update its FIB. In the case of flat routing, the local RIB will always be the single global RIB. From Figure 7, we can see that multi-level hierarchical RIBs effectively act as a multi-level cache, keeping the number of global RIB requests low as the workload increases.

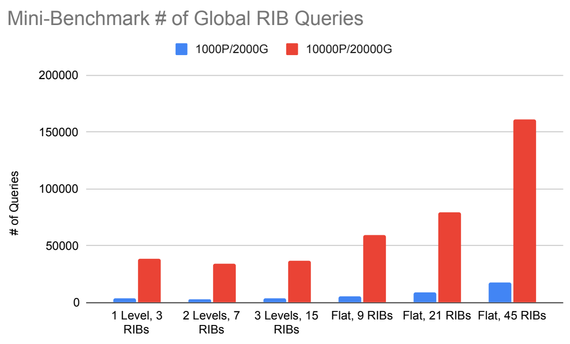

Figure 7. Number of Global RIB queries for multiple network topologies with an increasing number of multi-level RIBs compared to typical flat GDP routing with one Global RIB

5.3 Meta-DataCapsule

As discussed in section 3.2, we introduce the meta-DataCapsule to facilitate network communication and certificate verification. Our improvements provide most benefit when the distribution of DataCapsules across owners is non-uniform, such that a few owners own most of the DataCapsules. This allows us to aggregate certificates for these DataCapsules in the mDC. Figure 8 shows the speedups gained as more DataCapsules within a server come to be owned by a single owner. We still see a slowdown as more DataCapsules have to be sent across the network, but we no longer have to perform cryptographic verification for each one.

We can get similar speedups through batched certificate updating in a single packet and subsequent parallel processing by the network, although this has far more limited returns due to constraints on the network bandwidth and parallelism within RIBs.

6 Future Work

6.1 FIB Route Aggregation

In IP routing, RIBs can perform route aggregation to reduce the size of the routing table. In our case, this is not possible in an efficient manner because we must check individual valid bits and delegation certificates. We can, however, make use of aggregation in FIBs, which describe a subset of RIB entries. We can perform background route aggregation where we go through RIB entries and see which valid entries have the same next hop.

Note that we do not include delegation certificates in the FIB: if, for some reason, a certificate is not correct and the route is accessed through the FIB (so the error goes unnoticed at this RIB), it will still be caught by a later RIB. The amortized efficiency is still improved, assuming

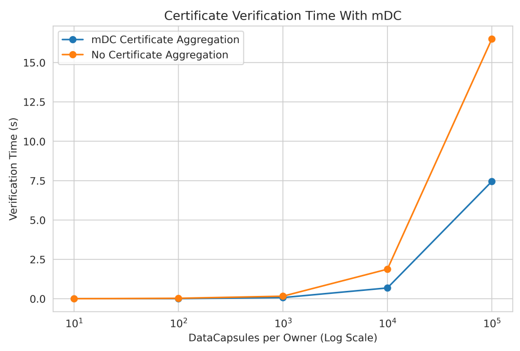

Figure 8. The line labeled "No Certificate Aggregation" shows the time taken to verify x DataCapsules under the current implementation, where every DataCapsule is sent to the RIB and its advertisement certificate is verified by the network. The second line shows the speed-up gained as these x verifications are replaced with a single certificate due to shared ownership.

that delegation certificate inconsistencies are not too common. This was not implemented in our existing code, but the potential improvements in latency warrant further work in this direction.

6.2 Real-world Evaluation

Our simulation-based evaluation may fail to account for the latencies and failures induced by the much more complex real-world network. We plan to migrate and further evaluate the algorithm's latency and reliability under real-world IOT, an example being FogROS.

6.3 Dealing with RIB Failure

Because crashes are inevitable and trust domains are modeled after real-world service providers, it is possible for RIBs to go offline or be moved. This alters the structure of an established hierarchy and requires the construction of new parent-child relationships, as well as updates that facilitate a consistent view of the network.

6.4 Heartbeat Messages

Our contribution's heartbeat mechanism could be developed further to have varying time intervals between heartbeat messages for different connections between RIBs. Specifically, longer intervals the further up the hierarchy would ensure that a short outage followed by a rapid recovery did not have to propagate all the way up the hierarchy.

6.5 Intra-TD and Distributed RIBs

Our current routing protocol focuses solely on inter-TD routing, which is unlikely to reflect large real-world TDs. Our protocols can be extended to support having multiple RIB servers within a single TD, as well as having multiple distributed nodes per RIB to support locality and larger routing tables. Using a distributed hash for RIB servers becomes more important as we go upwards in the hierarchy, where the size of the routing table naturally increases. Currently, we limit the size of all RIBs by having a fixed cache size and evicting invalid or least-recently-used entries, which can lead to cache misses when we are forced to evict valid routes.

7 Conclusion

We present a novel system of hierarchical routing information bases (RIBs) for the Global Data Plane (GDP), designed to optimize for reduced advertisement traffic, scalability with a large number of DataCapsule servers, and increased DataCapsule availability during server outages. We address the challenges of data-centric, location-agnostic guarantees, flat namespace, and security focus inherent to the GDP. Our key contributions include:

- Hierarchical RIBs: Building upon the nested trust domain structure of the GDP, we introduced a hierarchical system of RIBs that improves scalability and reduces routing overhead compared to a single global RIB.
- Source-based routing: We implemented a source-based routing protocol with valid bits and TD paths to efficiently route data packets across trust domains.
- Meta-DataCapsules: We introduced meta-DataCapsules on servers to efficiently detect changes in hosted DataCapsules and minimize advertising traffic.
- Failure protocols: We designed protocols for handling routing failures and re-routing packets to ensure data availability even when network errors occur.

By implementing this hierarchical routing system, the GDP can effectively scale to support a vast number of DataCapsules and ensure reliable data access for users across diverse trust domains. This paves the way for a decentralized, secure, and scalable data storage and retrieval infrastructure for the future.

8 Acknowledgements

We would like to thank Prof. Kubiatowicz for introducing us to the concept of GDP, its underlying principles, and the limitations of the current routing practices. He offered instrumental ideas in shaping our research direction. We would also like to thank Shubham Mishra and Kaiyuan Chen for providing continuous support to the research, helping refine our ideas and algorithms, and providing insightful suggestions during both ideation and implementation phases. Lastly, we thank the whole GDP team for spearheading the GDP framework.

References

[1] Alex Afanasyev, Lixia Zhang, Jeff Burke, Tamer Refaei, Lan Wang, and Beichuan Zhang. 2018. A Brief Introduction to Named Data Networking. (2018). https://named-data.net/wp-content/uploads/2019/01/NDN18.pdf

[2] Kaiyuan Chen, Ryan Hoque, Karthik Dharmarajan, Edith LLontop, Simeon Adebola, Jeffrey Ichnowski, John Kubiatowicz, and Ken Goldberg. 2023. FogROS2-SGC: A ROS2 Cloud Robotics Platform for Secure Global Connectivity. (2023). https://arxiv.org/pdf/2306.17157.pdf

[3] Tau Hu, Zehua Guo, Peng Yi, Thar Baker, and Julong Lan. 2018. Multi-controller Based Software-Defined Networking: A Survey. (2018). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8314783

[4] Shubham Mishra, Jiachen Yuan, and John Kubiatowicz. 2023. TrustNet: Trust-based routing for the Internet. (2023). Unpublished manuscript.

[5] 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)*. 1652–1663. https://doi.org/10.1109/ICDCS.2019.00164