GraphiteRouting: Name-based Hierarchical Routing for Internet-of-Things in Enterprise Environments

Qian Zhou and Fan Ye
Department of Electrical and Computer Engineering, Stony Brook University
Email: {qian.zhou, fan.ye}@stonybrook.edu

***Abstract*—Internet of Things in enterprise environments features large numbers of devices deployed in rooms, floors of possibly multiple buildings. Delivering user commands to control devices nearby and multiple hops away requires efficient and scalable routing in such environments. Existing work in ad-hoc, sensor or IoT network routing lacks good human accessibility and scalability. In this paper, we propose a peer-based protocol GraphiteRouting. All devices carry human-readable hierarchical string names for easy reference. It leverages devices' installation hierarchy for scalable hierarchical routing: most devices maintain only a few to dozens of routing entries for same-room devices, and a fraction of devices act as gateways for traffic from/to other rooms, floors or buildings. Also, it leverages users' operation patterns to less optimize infrequently used routes. Extensive analysis and performance evaluation on a 20-node testbed prove that GraphiteRouting is scalable: it has routing tables 10x$\sim$1000x smaller than those in peer-based flat routing; also, upon device joining/leaving, its routing entries converge in less than 5 s, and forwarding a user command over 8 hops costs less than 0.3 s.**

***Index Terms*—Peer-based Routing, Name-based Routing, Hierarchical Routing, Internet of Things**

I. Introduction

Internet of Things in enterprise environments features more than thousands of devices distributed in rooms, floors and possibly multiple buildings. A user issues a command to operate devices close by or multiple hops away, e.g., close/turn off all the "Things" (doors, windows, lights, etc.) in her office when leaving, or turn on the lights in the next aisle.

Routing is critical in correctly and efficiently delivering a command to the destination device. There is intensive routing work for the Internet, wireless sensor networks, mobile ad-hoc networks [1], [2], [3], and for IoT which focuses on energy efficiency [4], throughput [5], security [6], etc. However, a solution specifically devised for *enterprise IoT* deserves further effort, due to the context's unique characteristics.

First, though some IoT devices may be more resource-rich, a lot of constrained devices may not have direct Internet connectivity and they rely on peer-based routing to relay messages to/from users. Second, different from traditional ad-hoc networks that use numeric addresses to denote nodes, IoT features frequent human operations. Thus facilitating easy human access to discover, reference and control devices in manners convenient to humans is critical. Third, traditional ad-hoc network routing assumes "flat" traffic where any node may need to communicate with any other, but in IoT traffic is largely correlated with building structures. E.g., a user is mostly interacting with the devices installed in her room or floor. If routing paths are blindly established between all pairs of devices, it would incur prohibitive amounts of overhead, whereas most of which would be made in vain.

In this paper, we propose *GraphiteRouting*, a peer-based, name-based, hierarchical routing protocol leveraging IoT devices' installation hierarchy and users' operation patterns to achieve scalability. Each IoT device has a hierarchical name reflecting its building/floor/room of installation. Such names are easy for users to discover, remember, reference and access. By advertising names to neighboring devices, a fraction of devices find out their abilities to serve as room/floor/building gateways and they maintain routing entries for traffic across rooms/floors/buildings. And most devices are non-gateways which only maintain a few to dozens of routing entries for traffic within their rooms. A user command is hierarchically routed to the destination building first, then destination floor, room, and finally device. Besides, based on users' operation patterns, we use an ***intra-floor strict while inter-floor loose*** strategy that routing for traffic within a room or floor (which is frequent) is strongly optimized while that across floors or buildings (infrequent) is weakly optimized, to further reduce routing establishment overhead. It is named after graphite whose atoms have strong intra-layer forces while weak inter-layer forces. Our contributions are:

- We devise a hierarchical naming rule for IoT devices to facilitate easy human discovery, reference and access, without requiring a separate mechanism mapping human specifications to device numeric identifiers.
- We design a strategy for devices to dynamically establish a self-organized, hierarchical network based on their names. It leverages IoT devices' installation hierarchy to achieve hierarchical routing scalable to enterprise IoT.
- We strongly optimize frequent intra-room and inter-room routing, but not inter-floor or inter-building routing which occurs much more rarely. It reduces routing establishment overhead and further improves scalability.
- We implement our designs and conduct analysis and real experiments on a 20-node testbed. Using GraphiteRouting, one device needs at most dozens of routing entries regardless of the number or size of buildings. On the contrary, peer-based flat routing needs **10x$\sim$1000x** more entries. Besides, our solution has quick network convergence ($<$ **5** s when a device joins/leaves the network) and fast command forwarding ($<$ **0.3** s over 8-hop delivery).

II. Models and Assumptions

Node Types. Besides a backend server, two types of nodes are considered: subject devices and objects. A subject is a user who uses a subject device (e.g., smartphone) to access objects [7], [8], [9], i.e., IoT devices or "Things".

We argue that sufficient wall-powered objects (e.g., door locks, lights, HVAC) exist in enterprise environments. They have less concern about energy, and serve as routers. Battery-powered objects (e.g., temperature/smoke sensors) are hosts of routers and not involved in routing. In the remainder, objects refer to wall-powered ones unless otherwise specified.

Network Connectivity. We assume network connectivity exists among subject devices and objects in close proximity. This could be enabled by certain radio modes (e.g., WiFi ad-hoc, WiFi direct), and/or bridging devices that have multiple, possibly heterogeneous radios to relay messages. Objects are largely static once installed, but occasional, small-scale object additions/removals make the network topology low dynamic.

A. Properties of Internet-of-Things

1) Human operations are frequent. Human users cannot easily track numeric identifiers of objects. Instead, they are used to specifying an object to access with its installation location and device type, e.g., "living room, ceiling light".

2) Numerous objects are distributed in a building hierarchy. In a home environment there are usually dozens to hundreds of objects scattered in multiple rooms or a few floors; an enterprise can have up to tens of thousands of objects deployed in possibly multiple buildings, with many more floors per building and many more rooms per floor.

3) Command amounts vary inversely with target distances. Unlike Internet users who frequently perform remote access (e.g., stream movies from distant servers), IoT subjects are mostly interacting with nearby objects. i) Usually a subject is controlling those (e.g., lights, windows, air conditioner) in her current room. ii) Commands traveling across several rooms on the floor or across several floors, are less frequent but still common. E.g., before a manager leaves his office for conference room on the floor, he sends a command to boot the conference room computer/projector, such that when he arrives the equipment is ready; seeing from the 2nd floor that a visitor arrives, the secretary unlocks the building gate. iii) Remote commands crossing many floors or even targeting another building only happen occasionally.

III. Design Goals

Scalability. Routing table sizes and routing updating overhead (in terms of convergence time and message overhead) should be small enough such that the routing solution applies to large-scale IoT in enterprise environments.

Human Accessibility. Objects should be easily referenced, accessed by humans, without requiring a separate mechanism mapping human specifications to object numeric identifiers.

Self-Organization. The network should be self-organized, and dynamically updated upon object joining/leaving.

Non-Goals. We focus on IoT routing in enterprise environments, where objects are mostly installed indoor and human users send commands of small sizes to them. It is different from work [4], [10], [11] dealing with IoT of a wireless sensor network style where objects deployed outdoor with possibly constrained energy send large volumes of sensing data to gateways. Also, our work is peer-based, for robust and efficient routing in relative near ranges (e.g., < 10 hops). It is not for replacing infrastructure-based, long-distance routing.

IV. Routing Algorithm

Our routing algorithm is based on link-state (LS) routing. We choose LS other than distance-vector (DV) because it does not have the latter's count-to-infinity or slow convergence issues. However, LS originally may have too high message overhead to fit for an enterprise scale. Our design is name-based for good human accessibility, and it leverages IoT objects' installation hierarchy and subjects' operation patterns to achieve hierarchical routing scalable to enterprise IoT.

Bootstrapping. To join the system, a new subject or object must first register at the backend, getting its ID. An object additionally gets a hierarchical, human-readable string name according to its installation location and device type, with the format: Namespace/Building/Floor/Room/Type, e.g., SBU/EngBuilding/Floor1/Room217/Light1. The last segment can be any alias (e.g., "LightAboveWhiteBoard") as long as it is unique in that room. An object should get a new name from the backend if moved to another room/floor/building.

Roles. Each object has variable $roles$—a set tracking its role(s) in routing, which has one or more of the items below:

- **Internal object.** Each object has this role upon birth and will always do. It may gain one or more other roles.
- **Room-gateway router object.** A room-gateway routes commands from one room to another on the same floor.
- **Floor-gateway router object.** A floor-gateway routes commands from one floor to another in the building.
- **Building-gateway router object.** A building-gateway routes commands from one building to another.

Other Variables. Each object has $neighbors$ recording its 1-hop neighboring objects, and $roomMates$ recording other objects in its room. $roomMates$ constitute graph $roomGraph$ and the routes to them are stored in $roomRoutingTable$. If it is a room-gateway, it additionally has $floorMates$ recording other room-gateways on its floor. $floorMates$ constitute $floorGraph$ and the routes are in $floorRoutingTable$. $extra$ records objects in other floors or buildings.

Overview. Establishing routing has three steps: i) each object discovers neighbors and its own role; ii) each object runs LS algorithm to compute optimal paths to its $roomMates$; iii) each room-gateway runs LS algorithm to compute optimal paths to its $floorMates$. Most objects are non-gateways and do not perform the 3rd step. This reduces routing table sizes, updating overhead, and makes good scalability.

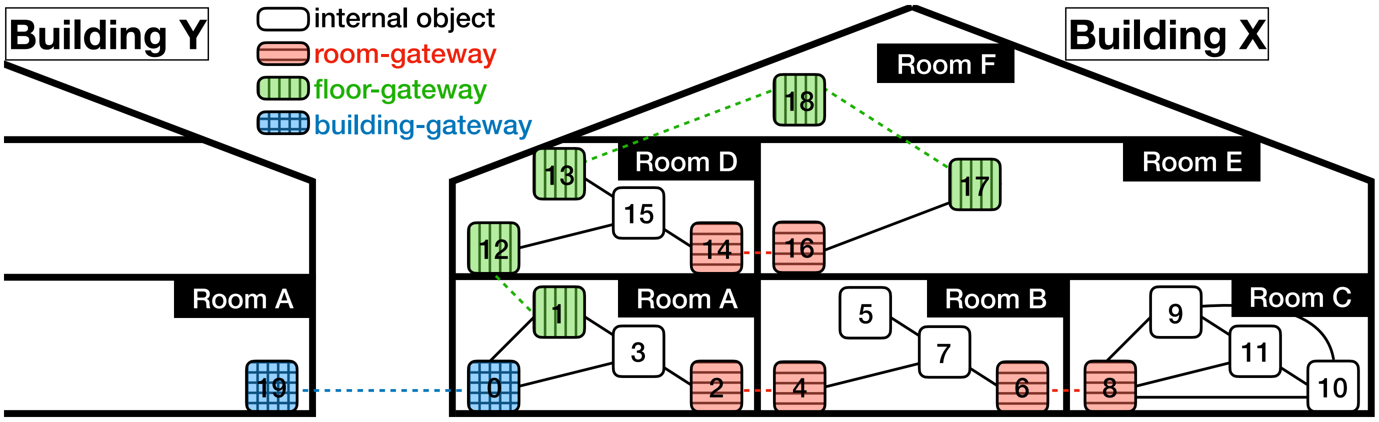


Fig. 1: A topology example of 20 objects deployed in two buildings, where a solid or dashed line denotes a wireless link. It is also the topology used by our testbed. Some objects are pure internal objects and have small routing tables; others are additionally room-/floor-/building-gateways and have slightly larger tables.

A. Neighbor and Role Updating

An object O periodically broadcasts HELLO messages for 1 hop (Algorithm 1: line 1–4). The message carries message type, sender ID, name, address (e.g., IP). t is a timestamp of message generation for ensuring that the latest information is used for routing updating; n is a nonce for duplicate detection.

Algorithm 1 Routing algorithm: neighbor and role updating

```
1:  procedure SEND HELLO
2:      $type \leftarrow$ ‘HELLO’
3:      broadcast HELLO: $\langle type, O, O.name, O.addr, t, n\rangle$
4:  end procedure

5:  procedure PROCESS HELLO($O', name', addr'$)
6:      $relation \leftarrow$ relation$(name', O.name)$
7:      $role \leftarrow$ relationToRole$(relation)$
8:      add $role$ to $O.roles$
9:      add $(O', name', addr')$ to $O.neighbors$
10: end procedure
```

When receiving a HELLO from O' (line 5–10), O compares its own name with O''s to get the maximum prefix length, and based on Tab. I it gets their location relationship and its role. E.g., in Fig. 1, Object 4 has name “SBU/BuildingX/Floor1/RoomB/Lamp”; it hears Object 2’s name “SBU/BuildingX/Floor1/RoomA/Lamp”, and finds 3 matching prefixes, which means they are on the same floor but different rooms. Then, a role is inferred from the relationship: since Object 4 hears an object in the next room, it finds its ability to serve as a bridge for traffic across the two rooms, i.e., it is a room-gateway. O’s $roles$ and $neighbors$ are updated.

B. Intra-Room Routing Updating

Now O has knowledge about its neighbors, it periodically generates its intra-room link state packets (ROOM_LSP) and propagates them within the room. The message carries O’s link states to *intra-room neighbors* (Algorithm 2: line 4–7). E.g., Object 4’s ROOM_LSP contains link 4-7. We do not enforce a metric for link cost, which can be hop count, expected transmission count [12], etc. $extra$ stores the information of inter-floor or inter-building neighbors (line 8–9), and will be used later in inter-floor or inter-building routing (Section IV-D).

TABLE I: Each maximum prefix length corresponds to a location relationship and a routing role.

Length	Location Relationship	Role
4	**intra-room**: O' and O in the same room	internal object
3	**inter-room**: different rooms, same floor	room-gateway
2	**inter-floor**: different floors, same building	floor-gateway
1	**inter-bldg**: in different buildings	bldg-gateway

Algorithm 2 Routing algorithm: intra-room routing updating

```
1:  procedure SEND ROOM_LSP
2:      $type \leftarrow$ ‘ROOM_LSP’
3:      $links \leftarrow \varnothing$, $extra \leftarrow \varnothing$
4:      for all $O_i \in O.neighbors$ do
5:          $relation \leftarrow$ relation$(O_i.name, O.name)$
6:          if $relation =$ ‘intra-room’ then
7:              add $(O, O_i, cost_i)$ to $links$
8:          else if $relation$=‘inter-floor’ or ‘inter-bldg’ then
9:              add $O_i.name$ to $extra$
10:         end if
11:     end for
12:     propagate ROOM_LSP: $\langle type, O, O.name, O.roles,$
        $links, extra, t, n\rangle$ within the room
13: end procedure

14: procedure PROCESS ROOM_LSP($O', name', roles',$
        $links', extra'$)
15:     add($O'$,$name'$,$roles'$,$cost'$,$extra'$) to $O.roomMates$
16:     add $links'$ to $O.roomGraph$, and update
        $O.roomRoutingTable$ using Dijkstra’s algorithm
17: end procedure
```

When receiving a ROOM_LSP from O' (line 14–17), O adds O''s information to its $roomMates$, and updates $roomGraph$ and $roomRoutingTable$. So far, it can route messages to other objects in its room along optimal paths.

C. Inter-Room Routing Updating

In this paper "inter-room" means across different rooms *on the same floor*. If O is a room-gateway, it periodically generates inter-room link state packets (FLOOR_LSP) which have a similar format as a ROOM_LSP, and propagates them within the floor. Unlike a ROOM_LSP, the message carries O's link states to *inter-room neighbors* and *intra-room room-gateways*. E.g., Object 4's FLOOR_LSP contains link 4-2 and 4-6. $extra$ in a FLOOR_LSP is a union of all the $extra$s of its $roomMates$, for inter-floor/building routing (Section IV-D).

If O is a room-gateway, when receiving a FLOOR_LSP from O', it adds O''s information to its $floorMates$, and updates $floorGraph$ and $floorRoutingTable$. So far, it can route messages to room-gateways in other rooms of its floor along optimal paths, while a pure internal object cannot.

Object 4's two routing tables are shown in Tab. II.

TABLE II: Object 4: $roomRoutingTable$ (left) & $floorRoutingTable$ (right)

Destination	Next Hop
5	7
6	7
7	7

Destination	Next Stop
2	2
6	6
8	6

D. Loose Inter-Floor/Building Routing

Unlike intra-room and inter-room routing, inter-floor and inter-building need no shortest paths maintained proactively. This is what we call "*intra-floor strict while inter-floor loose*".

Inter-Floor Routing. Inter-floor routing is different from the other three: floors are always named after regular numbers, while objects, rooms, buildings not necessarily. We use such numbers to devise a simple but effective routing strategy and a shortest path algorithm is not needed: to deliver a packet to the destination on Floor y, O on Floor x just checks if it or any of its roommates has a neighbor O_N on Floor x' (by checking roommates' $extra$ fields) such that $|y - x'| < |y - x|$; if yes, the packet is sent to O_N to vertically approach the destination; if no, O sends the packet to another room on the floor (via room-gateways) where the objects know such a neighbor O_N, and then vertical approach is performed.

Inter-Building Routing. If the destination is in a different building, O just checks if it or any of its roommates has a neighbor O_N in the target building, and forwards the packet to O_N if yes. Otherwise it sends the packet to another room of the floor/building (via room-/floor-gateways) for further search. This suffices routing to neighboring buildings.

Arguments. We use such loosely optimized strategies because: First, as mentioned in Section II-A, commands that need to reach a different building or cross many floors are rare, and always maintaining optimal routing paths for them would incur prohibitive amounts of overhead, whereas most of which would be made in vain. Instead, our strategies do not guarantee shortest paths, but they achieve effective inter-floor or building routing without needing extra link state exchange.

Second, occasionally commands need to travel far (e.g., to another building), and in such cases peer-based routing, even strongly optimized, would be too slow or unreliable because of too many hops. It is better to route the command to the target via infrastructures (access points, cables), or to its vicinity (e.g., the same room or floor), then our intra-room or inter-room routing can take the task over.

E. Dynamic Updating upon Object Joining/Leaving

By sending periodic HELLOs, ROOM_LSPs and FLOOR_LSPs, an object which joins the network will be detected by other objects. Those messages are all soft-state, so the entries of an object which leaves the network will be expired and then removed.

V. Forwarding Algorithm

A user command is hierarchically forwarded to its destination via routing table lookup: it reaches the destination building first, then destination floor, room, and finally device. An internal object depends on itself for optimal intra-room forwarding, and delegates to a gateway the traffic going out of the room to another room, floor or building.

As shown in Algorithm 3 (line 2–4), when receiving a command cmd, O extracts its destination $dest$. O executes cmd if it is $dest$, otherwise it finds its relationship with $dest$ by comparing their names. Depending on the relationship, one of the following forwarding strategies is used:

1) Intra-Room Forwarding. (line 5–8) If $dest$ is in the same room and exists in O's $roomMates$, O simply forwards cmd to it by $roomRoutingTable$ lookup.

2) Inter-Room Forwarding. (line 9–13) If $dest$ is in another room of the floor: i) if O is a room-gateway, it has $floorRoutingTable$ and can forward cmd towards the room-gateway in $dest$'s room; ii) otherwise O delegates cmd to a room-gateway in its room.

3) Inter-Floor Forwarding. (line 14–20) If $dest$ is on another floor of the building: i) (line 15–16) O checks if it or any of its roommates has a neighbor O_N which is on a floor closer to $dest$'s floor, and sends cmd to O_N if yes; ii) (line 17–18) otherwise O delegates cmd to a room-gateway in its room, to search O_N from other rooms on the floor.

4) Inter-Building Forwarding. If $dest$ is in another building, the process is similar to inter-floor forwarding, except that O_N is a neighbor in $dest$'s building.

Gateway Selection. When a pure internal object needs to use a room-gateway, it may find multiple candidates among its roommates and which is on the optimal path to the destination is unknown to it. We use a *hot-potato routing* [13] like strategy: the object sends the message to the closest room-gateway (e.g., with the minimum hop count). The chosen gateway, if not on the optimal path, will forward the message to the correct gateway. Besides, it will inform the object which gateway to use in the future for that destination.

Algorithm 3 Forwarding algorithm

```
1: procedure PROCESS COMMAND($cmd$)
2:     $dest \leftarrow cmd.destination$
3:     $relation = relation(dest, O.name)$
4:     if $dest = O.name$ then execute $cmd$
                                                  ▷ intra-room forwarding
5:     else if $relation$ = 'intra-room' then
6:         if $dest \in O.roomMates$ then forward $cmd$ towards $dest$
7:         else return ERROR
8:         end if
                              ▷ inter-room (but the same floor) forwarding
9:     else if $relation$ = 'inter-room' then
10:        if $O$ is 'room-gateway' and $\exists\ O_F \in O.floorMates$ has relation($O_F, dest$) = 'intra-room' then forward $cmd$ towards $O_F$
11:        else if $\exists\ O_R \in O.roomMates$ is 'room-gateway' then forward $cmd$ towards $O_R$
12:        else return ERROR
13:        end if
                                                  ▷ inter-floor forwarding
14:    else if $relation$ = 'inter-floor' then
15:        if $O$ is 'floor-gateway' and $O$ has neighbor $O_N$ approaching $dest$'s floor then forward $cmd$ to $O_N$
16:        else if $\exists\ O_R \in O.roomMates$ has neighbor $O_N$ approaching $dest$'s floor then forward $cmd$ towards $O_R$
17:        else if $O$ is 'room-gateway' and $\exists\ O_F \in O.floorMates$ and $O_F$ or $O_F.roomMates$ has neighbor $O_N$ approaching $dest$'s floor then forward $cmd$ to $O_F$
18:        else if $\exists\ O_R \in O.roomMates$ is 'room-gateway' then forward $cmd$ towards $O_R$
19:        else return ERROR
20:        end if
                   ▷ inter-bldg forwarding is omitted for space constraints
21:    end if
22: end procedure
```

VI. EVALUATION

We implement our routing and forwarding algorithms, and conduct analysis and real experiments on a tested consisting of 20 objects, each emulated by a Raspberry Pi 3. WiFi adhoc is used for communication between two nodes. As argued in Section II, wall-powered objects with needed radios serve as routers while resource-/power-constrained ones do not, thus using Pis can emulate networking and power aspects well.

Since the "graphite" network mirrors the building's construction hierarchy, a node's routing table size is about the number of nodes in the room it is in, plus (if it is a room-gateway) the number of rooms on the floor, which is always a very limited number regardless of the size of the whole building. We first theoretically analyze our room/floor routing table sizes; then we conduct real experiments using 20 objects deployed as Fig. 1 to test a representative case of several rooms/floors, and each intra-room network is 1~2 hops in diameter. In reality there can be more objects per room, but hop count would be limited, considering that mainstream radios (e.g., WiFi, Bluetooth, ZigBee) have reasonably far transmission distances (e.g., dozens of meters). So our testbed consisting of 20 objects is regarded sufficient.

We make HELLOs/ROOM_LSPs/FLOOR_LSPs sent with period $T_{hello} = 1$ s, $T_{room} = T_{floor} = 2$ s, and expired in $E_{hello} = 2$ s, $E_{room} = E_{floor} = 4$ s.

We find GraphiteRouting is scalable to enterprise environments: i) its routing table is small (dozens of entries) while peer-based flat routing is **10x~1000x**; ii) it has quick network convergence ($<$ **5** s when a device joins/leaves the network) and fast command forwarding ($<$ **0.3** s over 8-hop delivery).

A. Routing Table Size

Without loss of generality, we assume x (order of magnitude: $10^0 \sim 10^1$, i.e. one to dozens) buildings, averagely l ($10^0 \sim 10^1$) floors per building, m ($\sim 10^1$) rooms per floor, and n ($\sim 10^1$) objects per room. The numbers of room-/floor-/building-gateways in each room are all denoted as k ($\sim 10^0$).

In our system the number of routing entries kept by an object depends on its role(s): i) every object is an internal object and has $(n - 1)$ entries for its roommates; ii) if it also serves as a room-gateway, it additionally needs $(mk-1)$ entries for its floormates (i.e. same-floor room-gateways); iii) if it also serves as a floor-/building gateway, it needs one or a few entries for each of the floors/buildings it can reach in 1 hop; that number is usually a few and negligibly small. We summarize the table sizes with small numbers omitted:

TABLE III: Routing table sizes of GraphiteRouting

Role	Routing Table Size	In Our Testbed
Internal object	n ($\sim 10^1$)	Object 3 has 3 entries
Room-gateway	$n + mk$ ($\sim 10^1$)	Object 2 has 6 entries

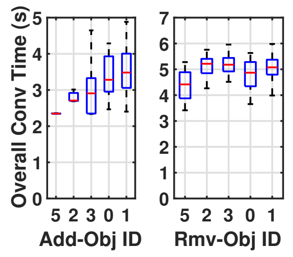
As shown, the table of a room-gateway and that of a pure internal object have the same magnitude ($\sim 10^1$). Both are significantly smaller than a flat routing table, which may have $xlmn$ ($10^2 \sim 10^4$) entries and be **10x~1000x** larger.

B. Message Overhead

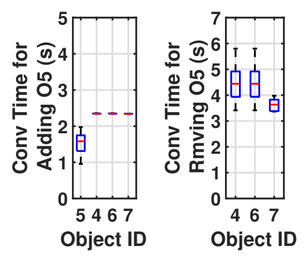
Theoretically, one HELLO is broadcast (1-hop only) by each object per T_{hello}; n ($\sim 10^1$) ROOM_LSPs are propagated within a room per T_{room}; mk ($\sim 10^1$) FLOOR_LSPs are propagated within a floor per T_{floor}. We regard such message overhead acceptably small, and the amount of traffic can be reduced if lower transmission frequencies are used (e.g., $T_{hello} = 10$ s, $T_{room} = 20$ s) at the expense of longer convergence time. In our implementation, HELLOs use simple broadcast. ROOM_LSPs, FLOOR_LSPs and CMDs have retransmission for reliable delivery. For each of the three, the retransmission rate found in our experiments is around 9.7%.

C. Routing Algorithm: Convergence Time

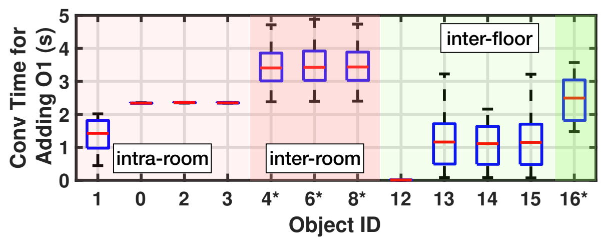
We test the convergence time (i.e. from an object's joining/leaving till the slowest affected object finishes updating its routing table) of five representative objects with different roles. They are: Object 5 (internal), 2 (room-gateway), 3 (internal), 0 (building-gateway), 1 (floor-gateway).



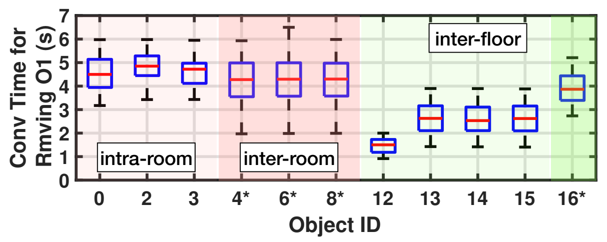
(a) Overall convergence time



(b) Convergence details for Object 5



(c) Convergence details when Object 1 joins. Floor-updating marked by ⋆



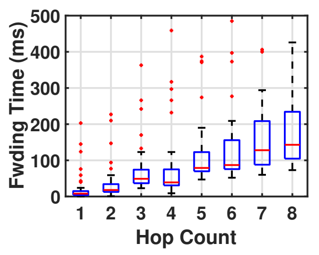
(d) Convergence details when Object 1 leaves. Floor-updating marked by ⋆

Fig. 2: Convergence time when different objects join/leave the network

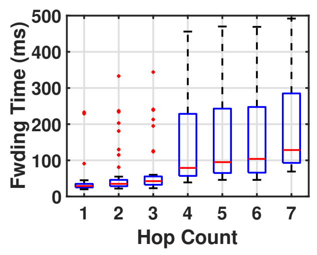
Fig. 2 (a) shows that all cases have fast convergence: < 5 s for addition and 6 s for removal. For addition, adding an internal object like Object 5 is the fastest since it only affects its roommates (details shown in (b)). A room-gateway (e.g., Object 2) is slower because of additional inter-room updating; a floor- or building-gateway (e.g., Object 1 or 0) is further slower due to updating in other floors or buildings besides of that in this room and floor (details shown in (c)(d)). Note that although Object 3 is a pure internal object, it has slower convergence than 5 because it happens to be a hub connecting Object 2 and 0, 1. Its existence makes Object 2 know the existence of a floor-gateway (Object 1) in the room, thus its presence/absence triggers Object 2's floor routing updating. On the other hand, convergence times for removal are less different among the five objects. It is because each case needs to wait for similar time for message expiration.

D. Forwarding Algorithm: Forwarding Time

We also evaluate the latency from a command's departure to arrival when using our forwarding algorithm. We test two complicated, multi-hop (8-hop and 7-hop respectively) forwarding cases. Case 1 is inter-building forwarding, from Object 10 to 19. Our forwarding algorithm finds the path: 10→8→6→7→4→2→3→0→19. Case 2 is inter-floor forwarding, from Object 0 to 18. Originally our algorithm gets: 0→1→12→15→13→18. To make it more challenging, we cut link 13-18, and find the algorithm succeeds in finding an alternative: 0→1→12→15→**14**→**16**→**17**→**18**. Fig. 3 presents the results for Case 1, and Case 2 without link 13-18.



(a) Command from Object 10 to 19



(b) Command from Object 0 to 18

Fig. 3: Command forwarding latency

Both cases achieve quick forwarding, usually < 0.3 s, and the median time costs are < 0.15 s. Note that the same hop count in the two figures corresponds to totally different objects which are deployed in different locations, and it is normal that their arrival times differ.

VII. Discussion

Link Cost. The link cost metric can affect directly the routing decisions. Hop count is one option. If more dynamic changes turn out significant, metrics accounting for them (e.g., expected transmission count (ETX) [12] can also be used.

Gateway Reduction. Due to objects' high deployment densities and radios' long transmission distances, it is common that many objects can hear objects in other rooms/floors, thus are able to act as gateways. However, GraphiteRouting wants most objects to be pure internal nodes and have small routing tables. The number of gateways per room for reaching the same external destination should be limited to a few, via e.g., gateway election, which is out of the scope.

Event-Driven Updating. Periodic updating is used in our design and implementation for its simplicity and robustness. An event-driven scheme that an object sends LSPs immediately when detecting a link state change, can result in faster updating and less traffic. The two can be used in combination.

VIII. Related Work

Ad-hoc Routing. DSR [1] is on-demand source routing which needs a message to carry the whole route with it. DSDV [2] relies on routing tables maintained by intermediate nodes and uses sequence numbers to avoid routing loops. Its periodic updating makes it unsuitable for highly dynamic networks. Like DSR, AODV [3] is an on-demand routing protocol; however, it maintains routing tables on nodes. They work for "flat" traffic where any node may need to communicate with any other, thus unscalable in enterprise IoT due to their significant unnecessary overhead.

Named-data Routing. NLSR [14] is named-data link state routing. Unlike IP-based, it propagates reachability information denoted by name prefixes instead of IP prefixes. Our naming rule uniquely mirrors buildings' construction hierarchy, and realizes scalable routing in enterprise IoT.

IoT Routing. Multiple IoT routing solutions with various goals have been proposed. In [15], [10], [4], IoT of a wireless sensor network style is targeted, and energy efficient routing is studied. In [11], content aggregation is performed during routing to alleviate congestion and reduce latency; Lei et al. [5] apply network coding techniques to named data networks (NDN) [16] for high throughput in 5G IoT. In [17], path survivability is considered in routing decision making. Real-time routing is studied in [18], and secure routing in [6]. They do not target the scalability issue in enterprise IoT.

Afanasyev et al. [19], [20] improve NDN routing scalability: instead of expensively maintaining forwarding information for all prefixes, forwarders only do that for a subset of prefixes to which other producers delegate their namespaces. In [21], to eliminate such maintenance, forwarders make on-demand routing decisions. Laser [22] has a hierarchical network: cluster heads maintain routes to each other, and are used by their cluster members for inter-cluster communication.

Though using hierarchy for high scalability is not new, ours is specifically devised for indoor enterprise IoT environments and uniquely congruent with objects' installation hierarchy. Thus objects' roles can be automatically, reasonably determined instead of manually appointed. Besides, subjects' operation patterns (e.g., intra-room operations are frequent, inter-building ones not) are leveraged to reduce routing establishment overhead and further improve scalability.

IX. CONCLUSION

In this paper we describe the design, implementation and evaluation of GraphiteRouting, a peer-based, name-based, hierarchical routing protocol leveraging building hierarchy. Most devices maintain routing entries for same-room devices only, and a fraction of them are gateways for traffic from/to outside of the room. It has good human accessibility, fast routing table updating (< 5 s for convergence), quick command forwarding (< 0.3 s over 8 hops), and is scalable (at most dozens of routing entries per node) to enterprise IoT.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation under grant number 1652276.

REFERENCES

[1] D. B. Johnson, D. A. Maltz, J. Broch *et al.*, "Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks," *Ad hoc networking*, vol. 5, pp. 139–172, 2001.

[2] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *ACM SIGCOMM computer communication review*, vol. 24, no. 4. ACM, 1994, pp. 234–244.

[3] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," Tech. Rep., 2003.

[4] F. Al-Turjman, "Cognitive routing protocol for disaster-inspired internet of things," *Future Generation Computer Systems*, vol. 92, pp. 1103–1115, 2019.

[5] K. Lei, S. Zhong, F. Zhu, K. Xu, and H. Zhang, "An ndn iot content distribution model with network coding enhanced forwarding strategy for 5g," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2725–2735, 2017.

[6] D. Airehrour, J. Gutierrez, and S. K. Ray, "A lightweight trust design for iot routing," in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2016, pp. 552–557.

[7] Q. Zhou, M. Elbadry, F. Ye, and Y. Yang, "Flexible, fine grained access control for internet of things," in *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2017, pp. 333–334.

[8] ——, "Heracles: Scalable, fine-grained access control for internet-of-things in enterprise environments," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1772–1780.

[9] Q. Zhou and F. Ye, "Apex: automatic precondition execution with isolation and atomicity in internet-of-things," in *Proceedings of the International Conference on Internet of Things Design and Implementation*. ACM, 2019, pp. 25–36.

[10] J. Shen, A. Wang, C. Wang, P. C. Hung, and C.-F. Lai, "An efficient centroid-based routing protocol for energy management in wsn-assisted iot," *IEEE Access*, vol. 5, pp. 18 469–18 479, 2017.

[11] Y. Jin, S. Gormus, P. Kulkarni, and M. Sooriyabandara, "Content centric routing in iot networks and its integration in rpl," *Computer Communications*, vol. 89, pp. 87–104, 2016.

[12] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wireless networks*, vol. 11, no. 4, pp. 419–434, 2005.

[13] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in ip networks," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1. ACM, 2004, pp. 307–319.

[14] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: named-data link state routing protocol," in *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 2013, pp. 15–20.

[15] C. Wu, D. Gunatilaka, A. Saifullah, M. Sha, P. B. Tiwari, C. Lu, and Y. Chen, "Maximizing network lifetime of wirelesshart networks under graph routing," in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2016, pp. 176–186.

[16] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[17] M. Elappila, S. Chinara, and D. R. Parhi, "Survivable path routing in wsn for iot applications," *Pervasive and Mobile Computing*, vol. 43, pp. 49–63, 2018.

[18] C. Wu, D. Gunatilaka, M. Sha, and C. Lu, "Real-time wireless routing for industrial internet of things," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2018, pp. 261–266.

[19] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang, "Map-and-encap for scaling ndn routing," *Univ. California at Los Angeles, Los Angeles, CA, USA, Rep. NDN-0004*, 2015.

[20] ——, "Snamp: Secure namespace mapping to scale ndn forwarding," in *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2015, pp. 281–286.

[21] O. Ascigil, S. Rene, I. Psaras, and G. Pavlou, "On-demand routing for scalable name-based forwarding," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*. ACM, 2018, pp. 67–76.

[22] T. Mick, R. Tourani, and S. Misra, "Laser: Lightweight authentication and secured routing for ndn iot in smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 755–764, 2017.