Graph-based Namespaces and Load Sharing for Efficient Information Dissemination in Disasters

Mohammad Jahanian *(University of California, Riverside); Jiachen Chen (WINLAB, Rutgers University); K. K. Ramakrishnan (University of California, Riverside)*

ICNP 2019

Information dissemination in disasters

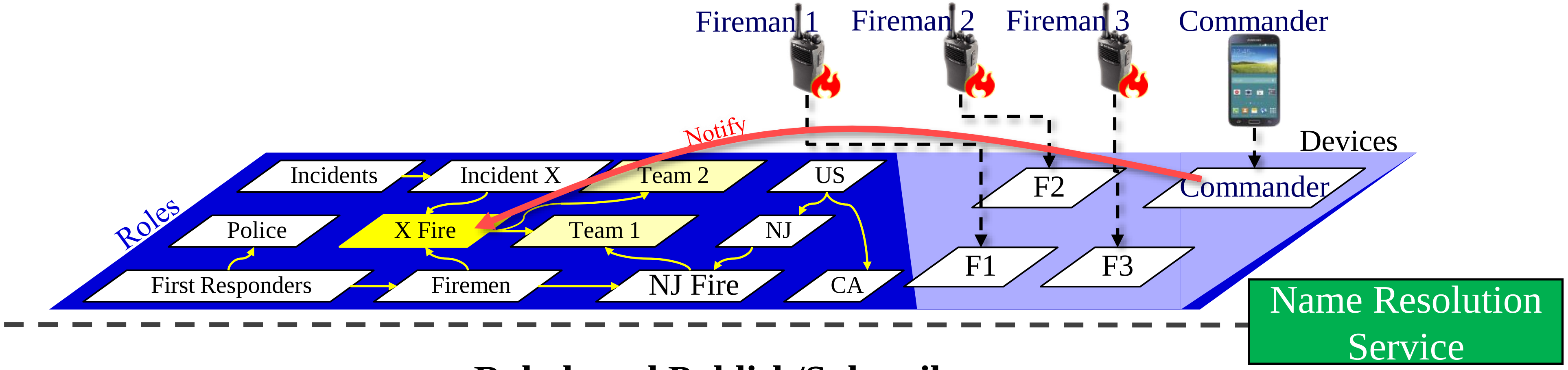
- Communication and information dissemination key in disaster management
    - Many-to-many, according to roles (e.g., instruction to all firefighters)
    - Many actors interacting with complex & dynamic relationships
    - Non-uniform demand: traffic concentration and congestion
    - Timeliness, relevance, coverage are important requirements
        - Timeliness: Information delivered in a timely manner
        - Relevance: Information delivered to the relevant people
        - Coverage: Information delivered to everyone who needs it

Information dissemination in disasters

- Communication and information dissemination key in disaster management
  - Many-to-many, according to roles (e.g., instruction to all firefighters)
  - Many actors interacting with complex & dynamic relationships
  - Non-uniform demand: traffic concentration and congestion
  - Timeliness, relevance, coverage are important requirements
- **POISE**: Information dissemination for disaster management

Information dissemination in disasters

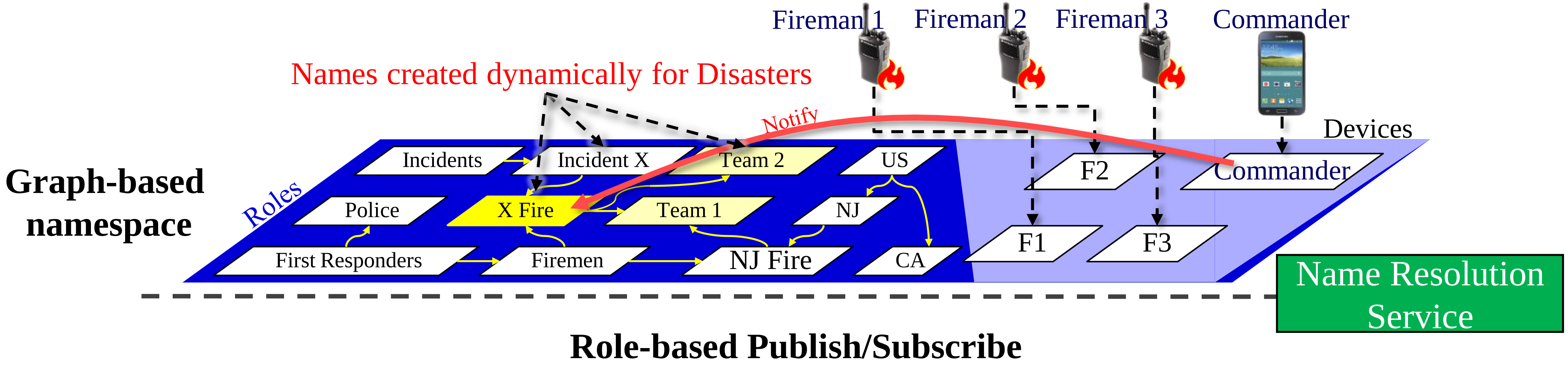
- Communication and information dissemination key in disaster management
  - Many-to-many, according to roles (e.g., instruction to all firefighters) → Role-based pub/sub
  - Many actors interacting with complex & dynamic relationships
  - Non-uniform demand: traffic concentration and congestion
  - Timeliness, relevance, coverage are important requirements
- **<u>POISE</u>**: Information dissemination for disaster management enabling role-based pub/sub



Role-based Publish/Subscribe

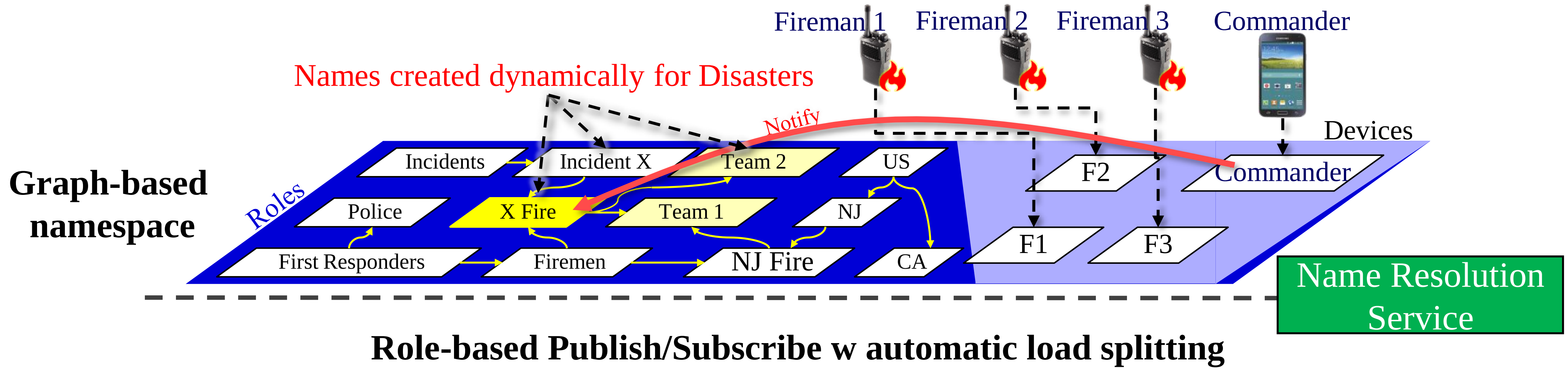
Information dissemination in disasters

- Communication and information dissemination key in disaster management
  - Many-to-many, according to roles (e.g., instruction to all firefighters) → Role-based pub/sub
  - Many actors interacting with complex & dynamic relationships → Graph-based namespace
  - Non-uniform demand: traffic concentration and congestion
  - Timeliness, relevance, coverage are important requirements
- **<u>POISE</u>**: Information dissemination for disaster management enabling role-based pub/sub, supporting graph-based namespaces



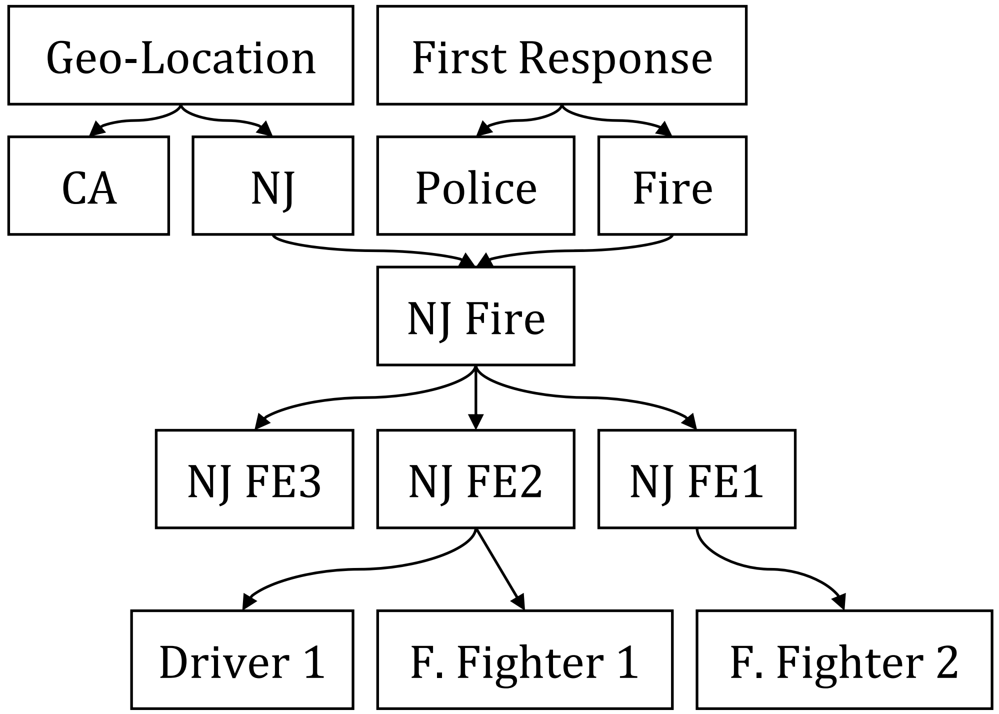
Information dissemination in disasters

- Communication and information dissemination key in disaster management
  - Many-to-many, according to roles (e.g., instruction to all firefighters) → Role-based pub/sub
  - Many actors interacting with complex & dynamic relationships → Graph-based namespace
  - Non-uniform demand: traffic concentration and congestion → Load sharing and splitting
  - Timeliness, relevance, coverage are important requirements
- **POISE**: Information dissemination for disaster management enabling role-based pub/sub, supporting graph-based namespaces, with automatic load splitting



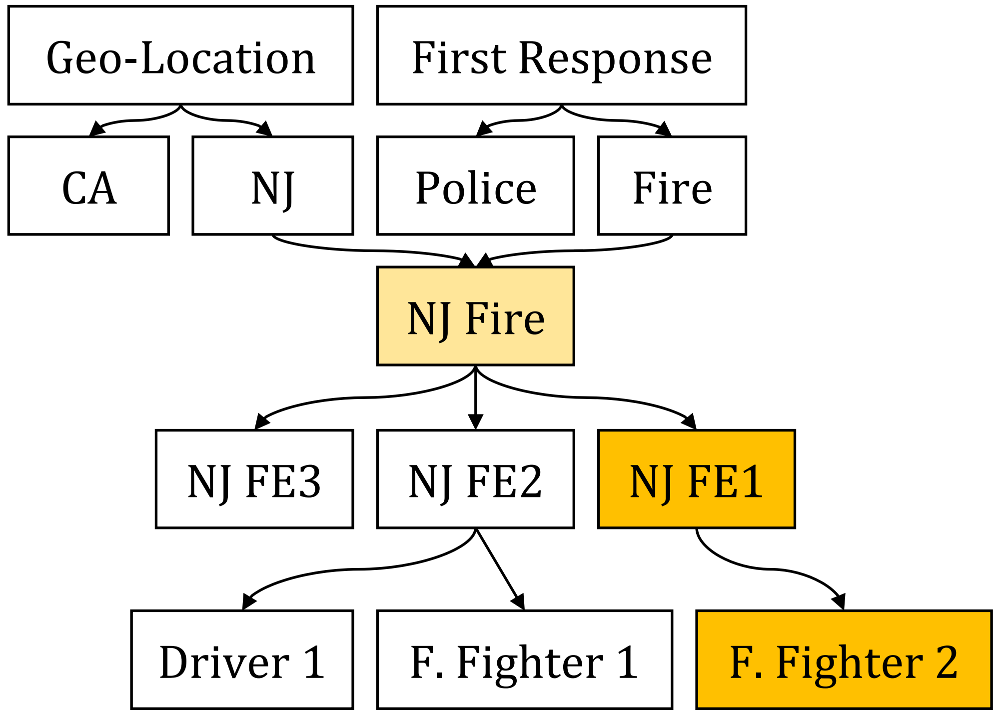
Graph-based namespace for disaster management

- Information flow organization
- Multi-dimensional structure
- Nodes are names. Edges are name relationships



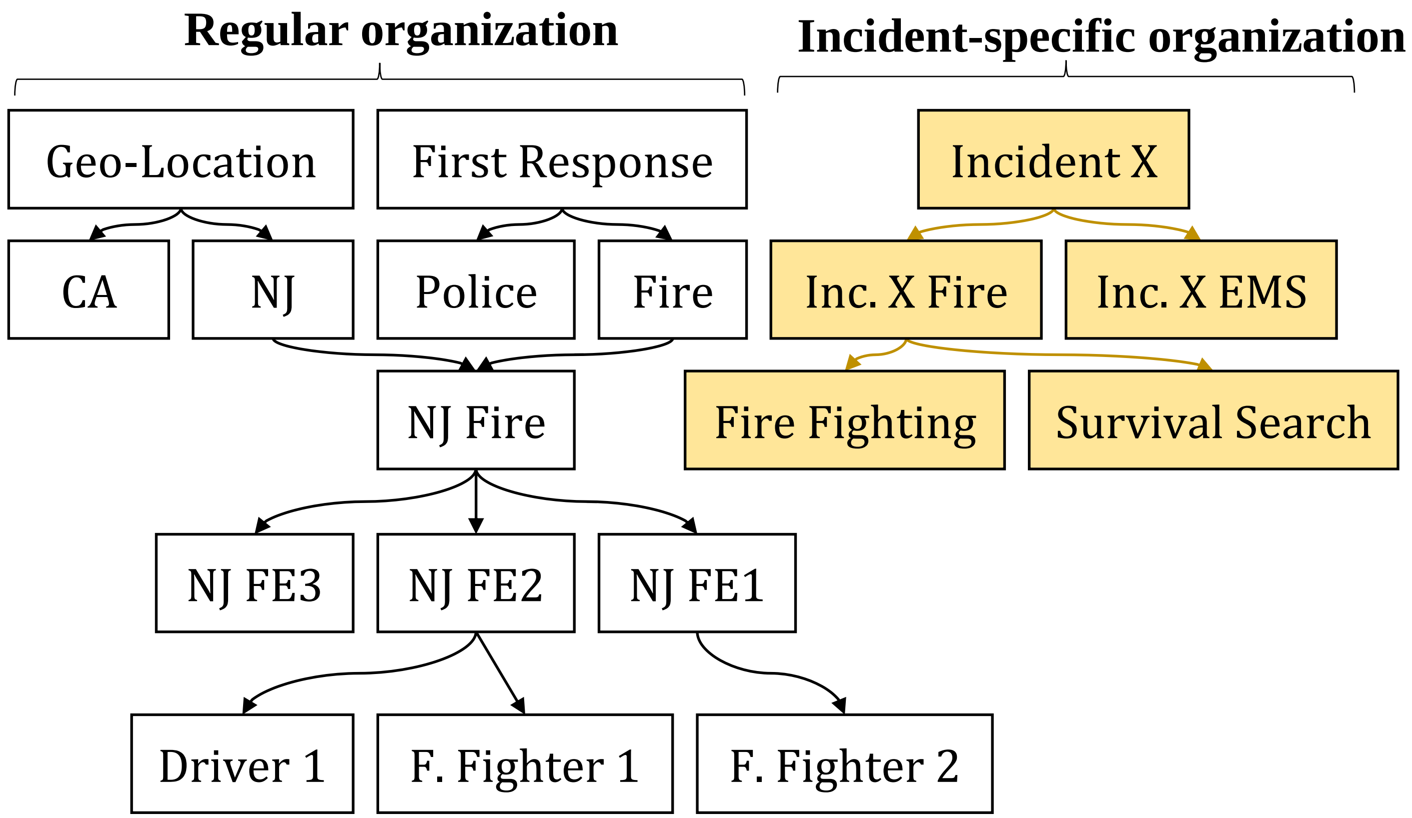
Graph-based namespace for disaster management

- Information flow organization
- Multi-dimensional structure
- Nodes are names. Edges are name relationships
- “NJ Fire” denotes all fire-related tasks in New Jersey
- “NJ FE1” (NJ fire engine 1) is a higher-level authority than “F.Fighter2” (fire fighter 2)



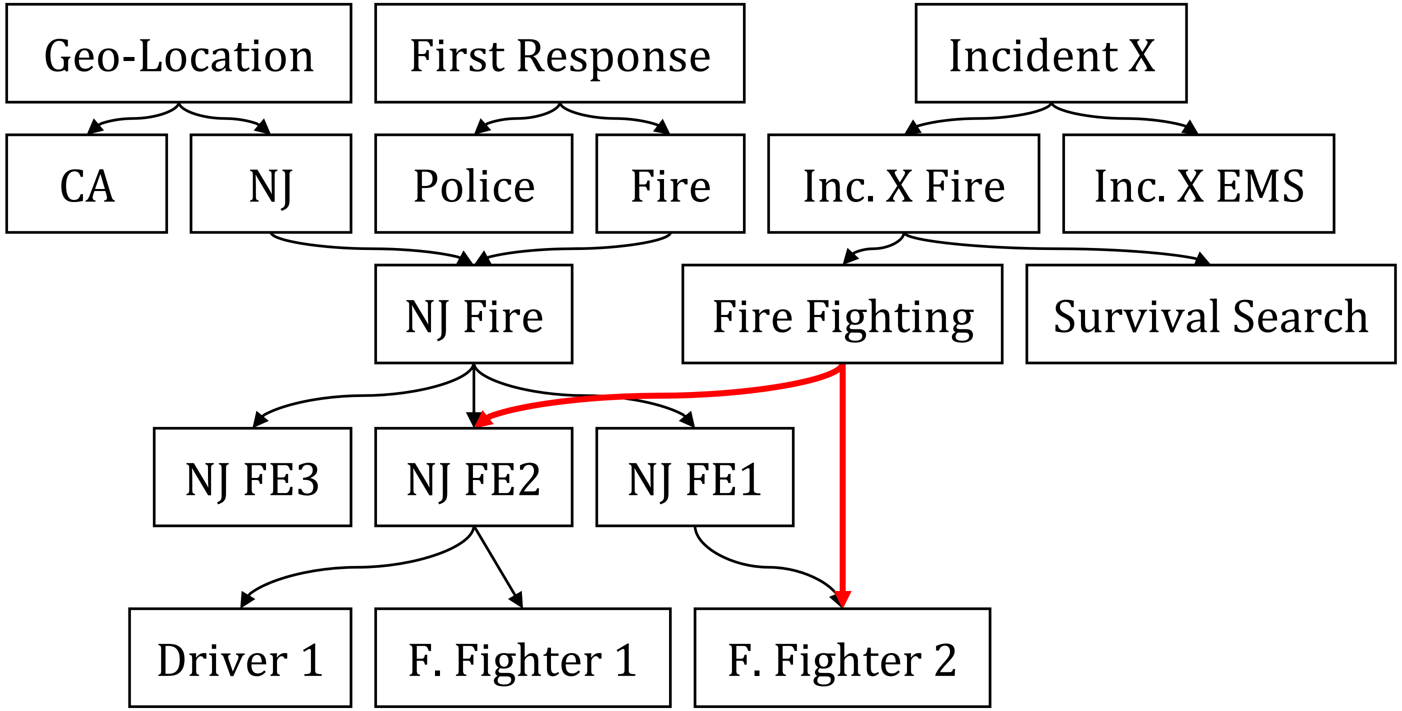
Graph-based namespace for disaster management

- Many different sub-namespaces
  - Organizations, incidents
- New names/roles for an incident can be added
  - Incident X sub-namespace added



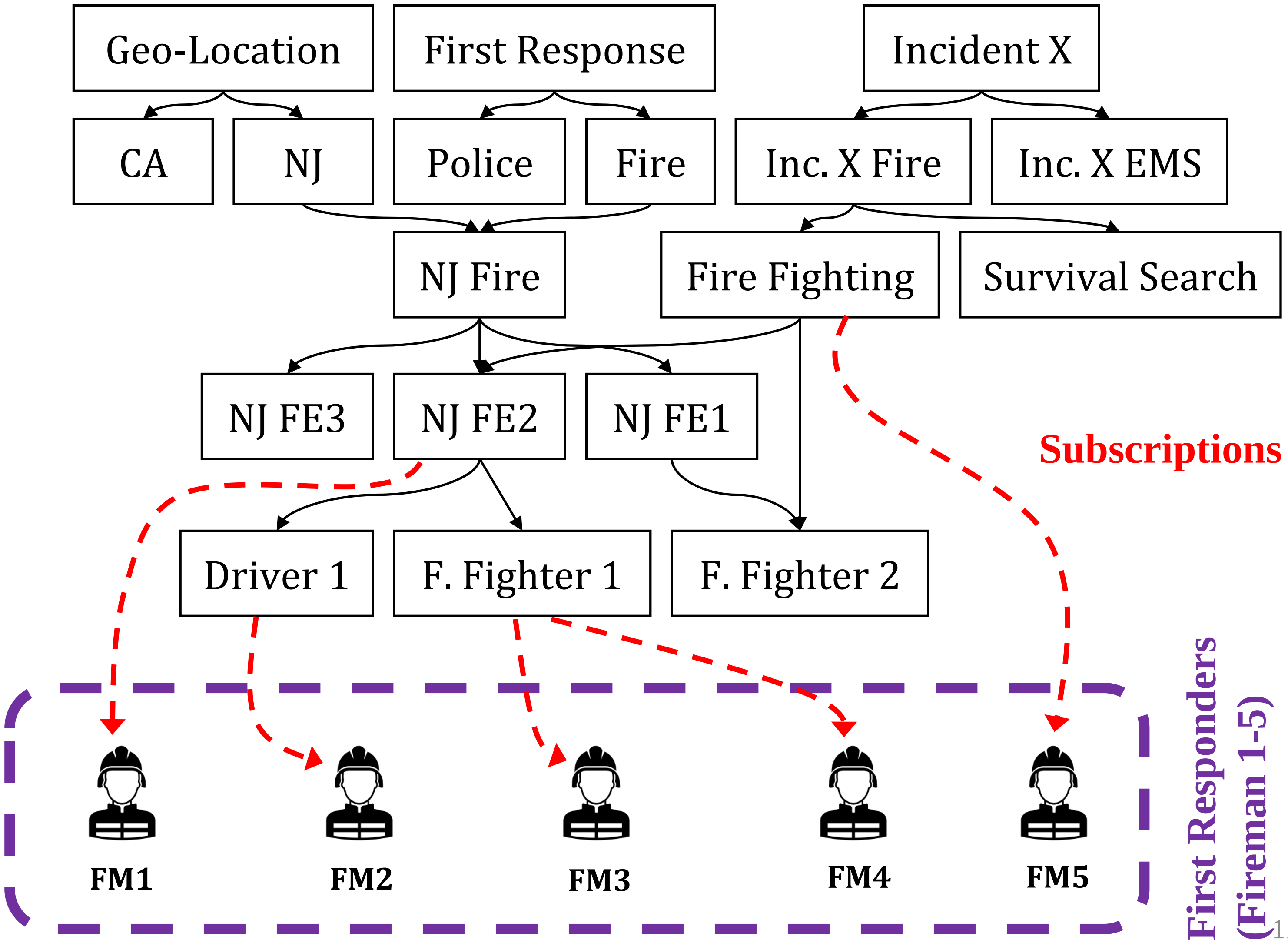
Graph-based namespace for disaster management

- Many different sub-namespaces
  - Organizations, incidents
- New names/roles for an incident can be added
  - Incident X sub-namespace added
- Edges can be added/removed
  - "NJ FE2" and "F. Fighter 2" dispatched for "Fire Fighting" in Incident X



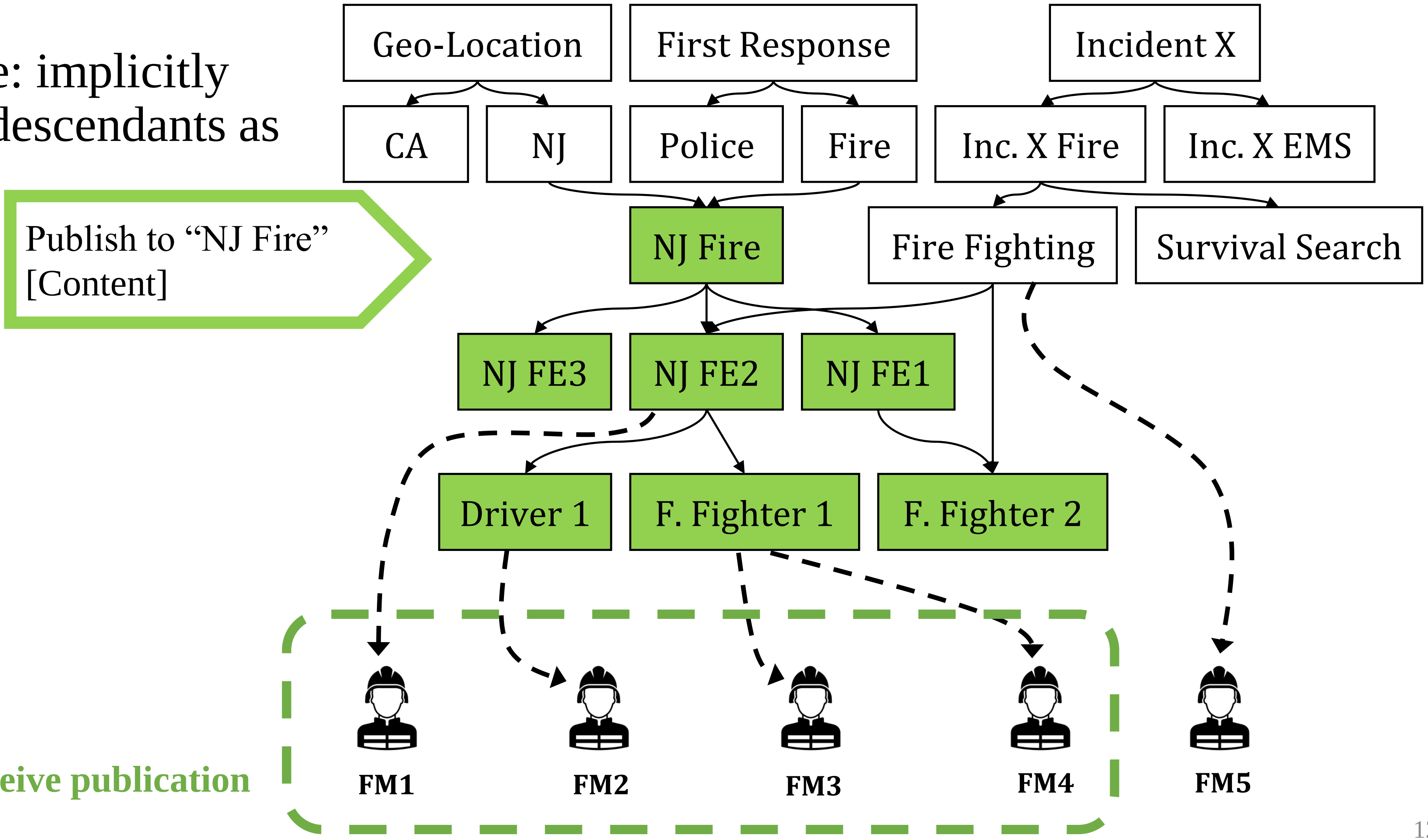
Graph-based namespace for disaster management

- First responders subscribe to ("listen to") names
  - Roles they are associated with
    - FM3 subscribe to/responsible for "F.Fighter 1"
  - At appropriate level of granularity
  - They will receive publications to those name whenever published
- Incident commanders (or any users) "publish" to names
  - Publications to "F. Fighter 1" will reach FM3 and FM4
- Recipient-based pub/sub (CNS[ICN'16]), but w graphs



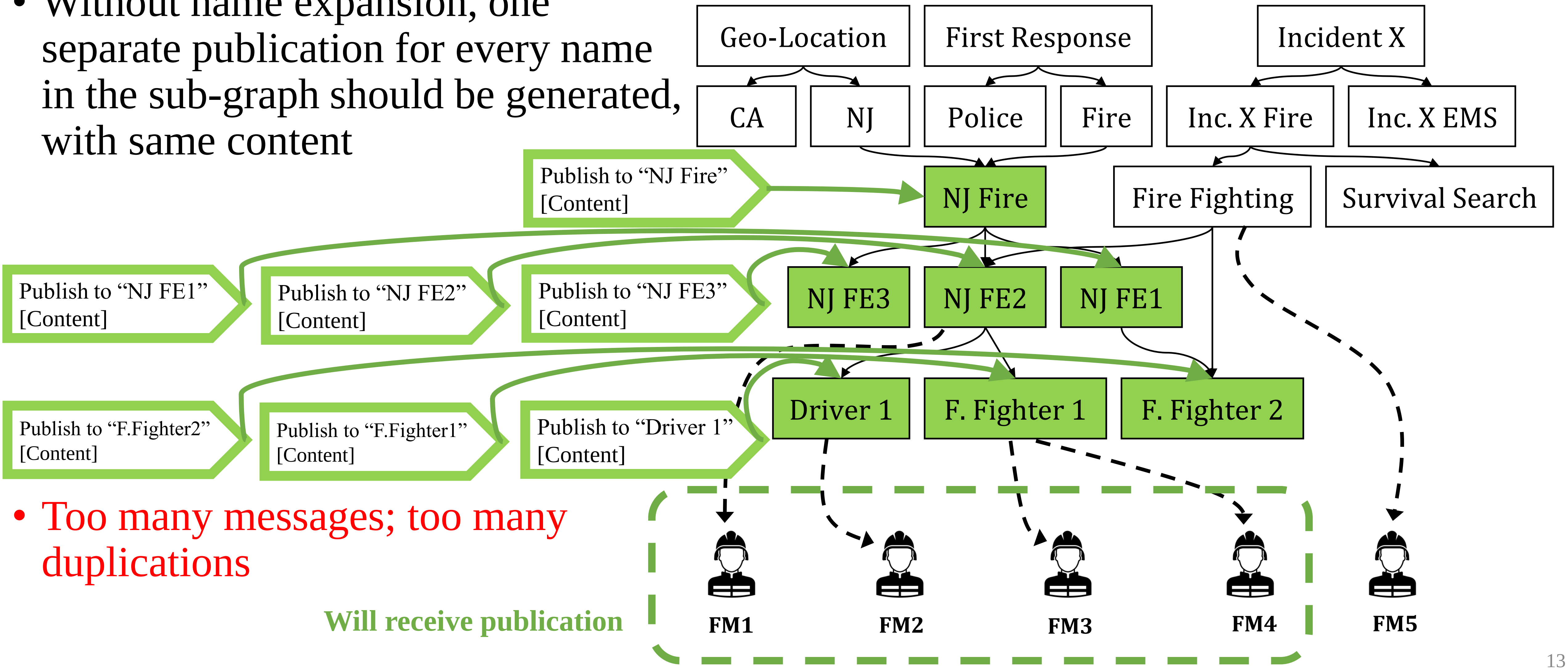
Graph-based namespace for disaster management

- Name expansion
  - Publishing to a name: implicitly publishing to all its descendants as well



Graph-based namespace for disaster management

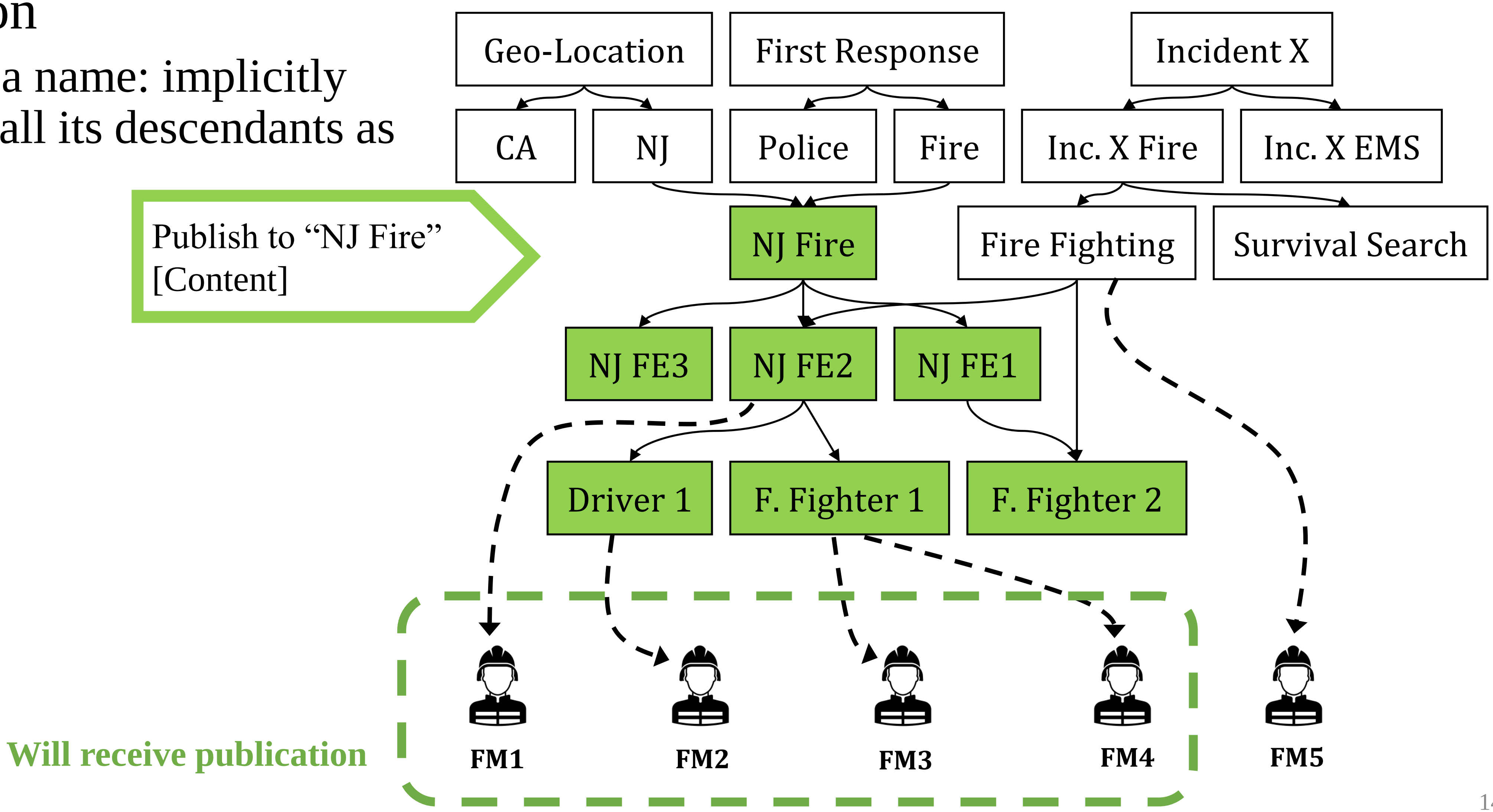
- Without name expansion, one separate publication for every name in the sub-graph should be generated, with same content



- Too many messages; too many duplications

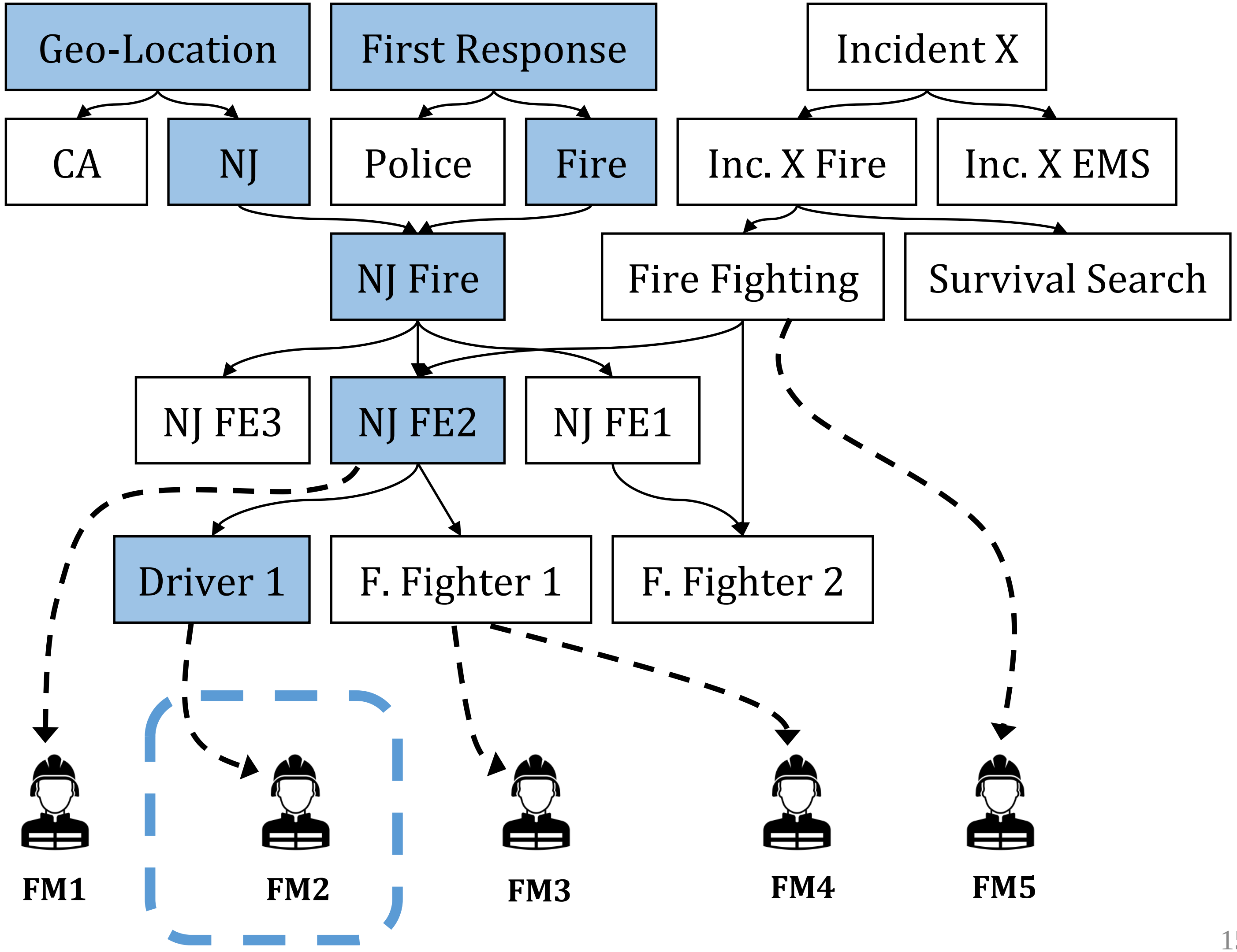
Graph-based namespace for disaster management

- Name expansion
  - Publishing to a name: implicitly publishing to all its descendants as well



Graph-based namespace for disaster management

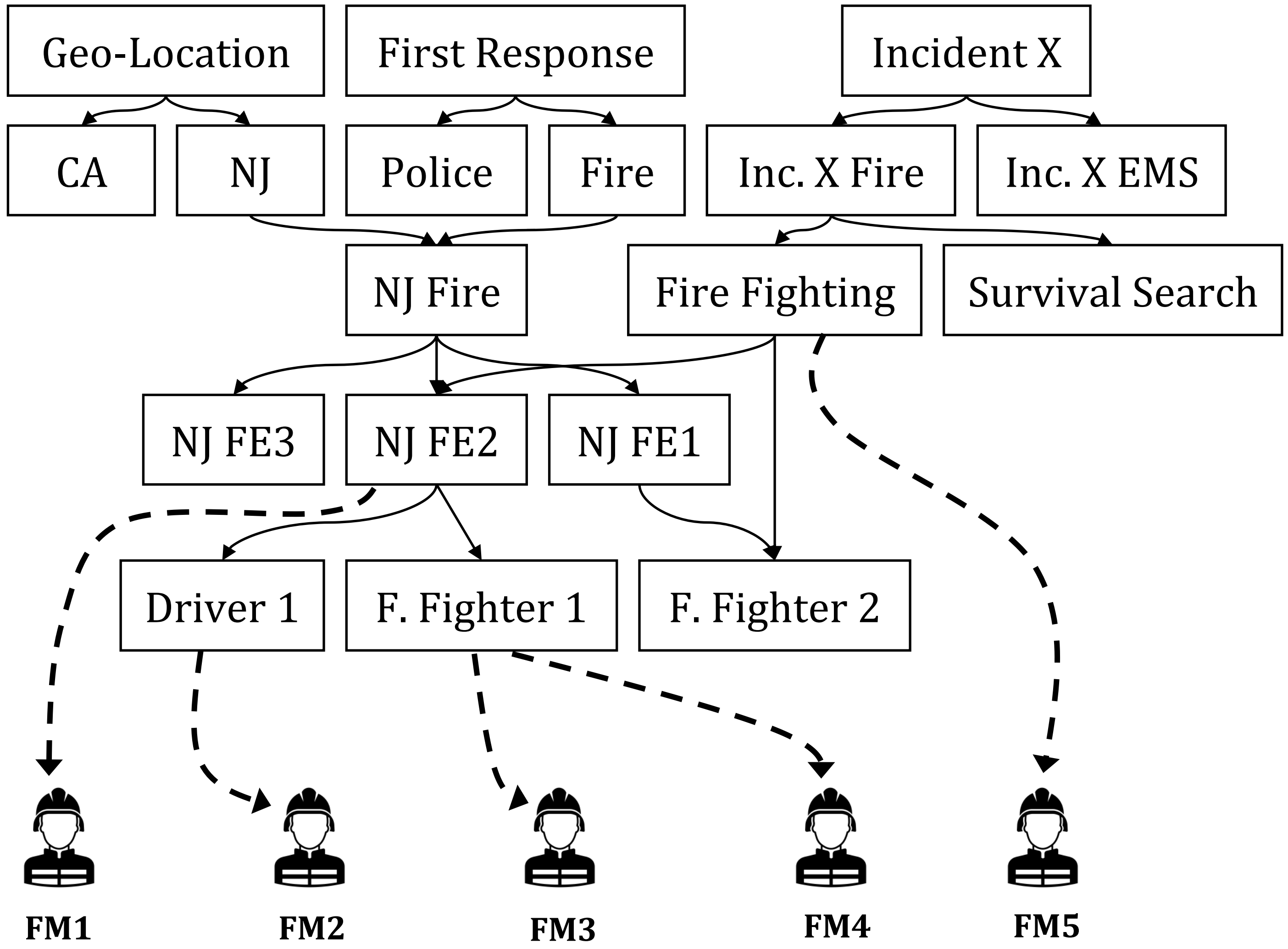
- Name expansion
  - Publishing to a name: implicitly publishing to all its descendants as well
  - Subscribing to a name: implicitly subscribing to all its ancestors as well



Subscribe to Driver 1

Graph-based namespace for disaster management

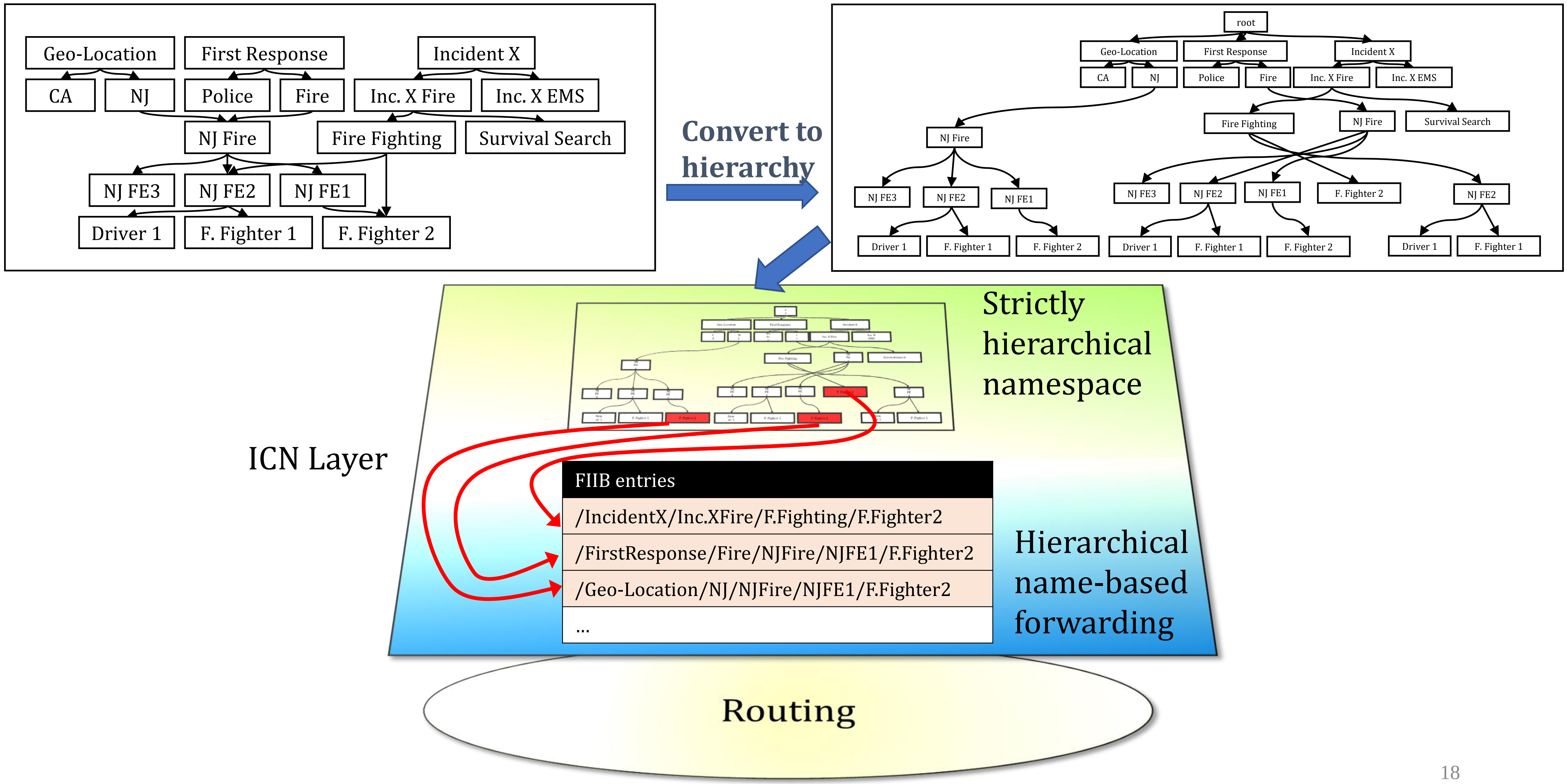
- Name expansion
  - Publishing to a name: implicitly publishing to all its descendants as well
  - Subscribing to a name: implicitly subscribing to all its ancestors as well
  - Greatly decreases subscription & publication messages (network resources and user load)
  - Need to support in the network



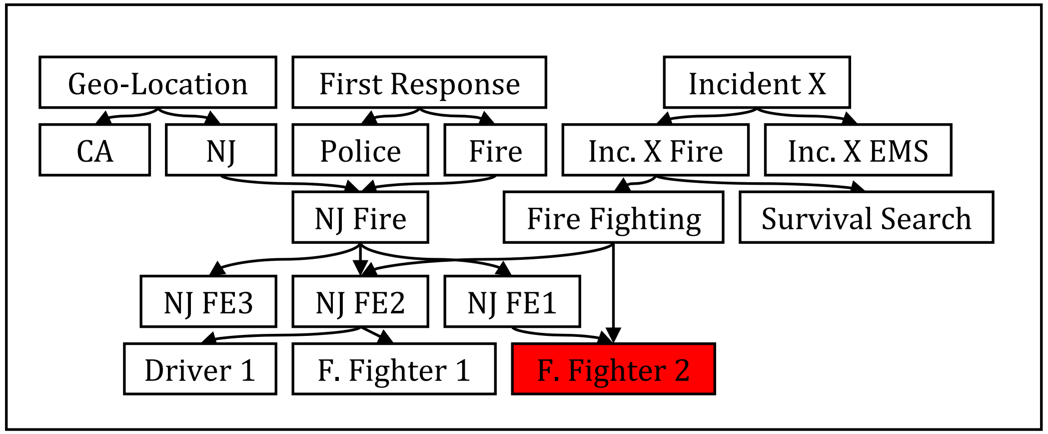
Support graph-based namespaces in the network

- Need support in network (multicast) for efficient delivery
- IP multicast is feasible but has issues
  - Flat IP address space, cannot capture multicast-group inter-relationship
- Information-Centric Networking (ICN) enables name-based multicast
  - However, state-of-the-art supports hierarchical naming in the network: Named Data Networking (NDN)

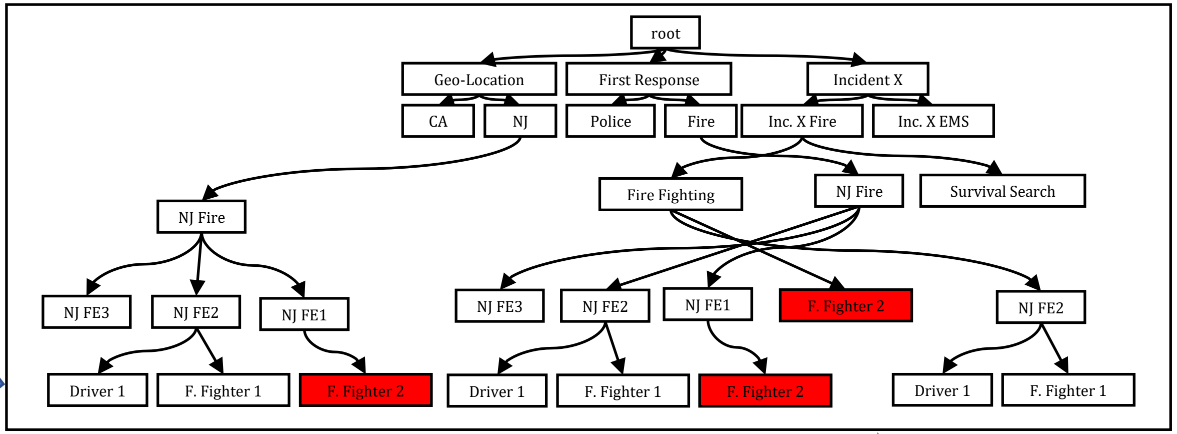
Graph-based pub/sub using traditional ICN



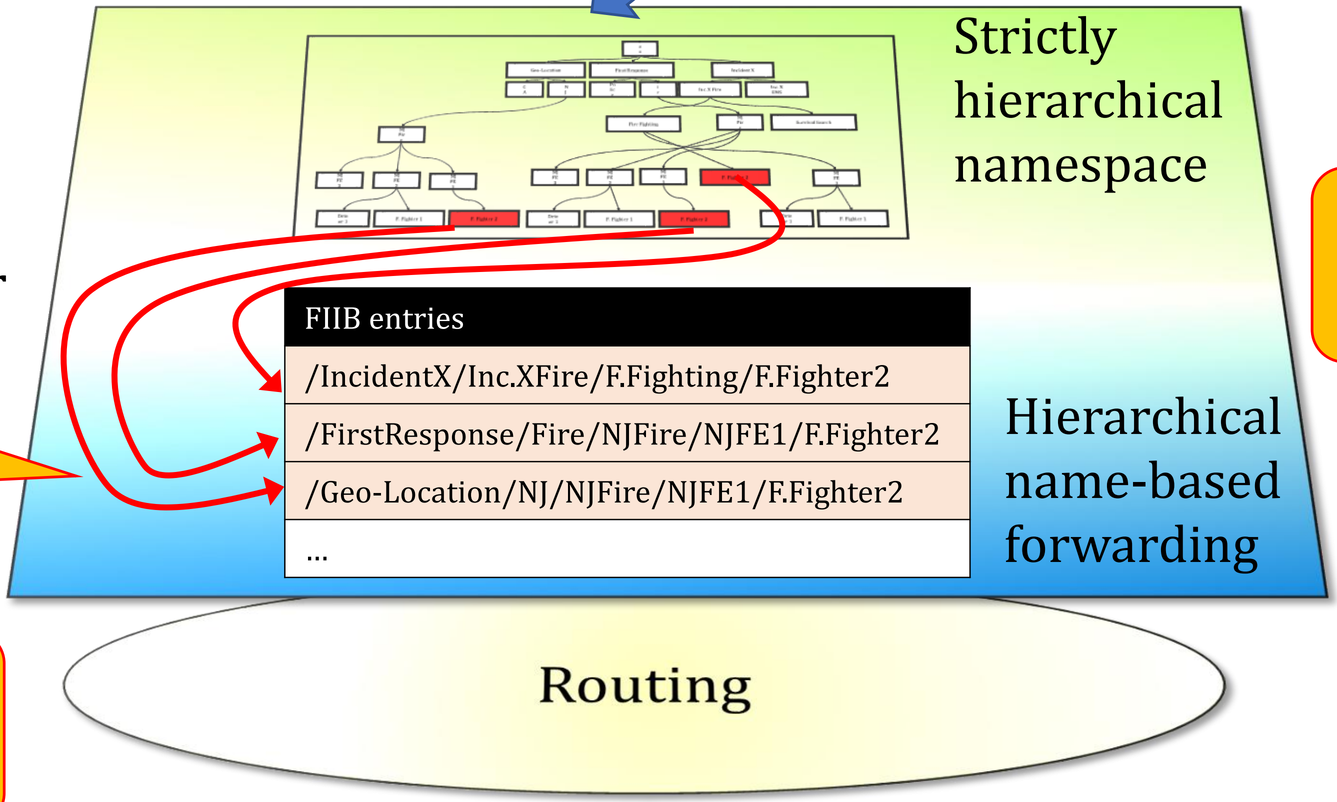
Graph-based pub/sub using traditional ICN



Convert to hierarchy



ICN Layer



"F. Fighter 2" appears three times in the hierarchical equivalent

"F. Fighter 2" appears as three separate entries in the FIB

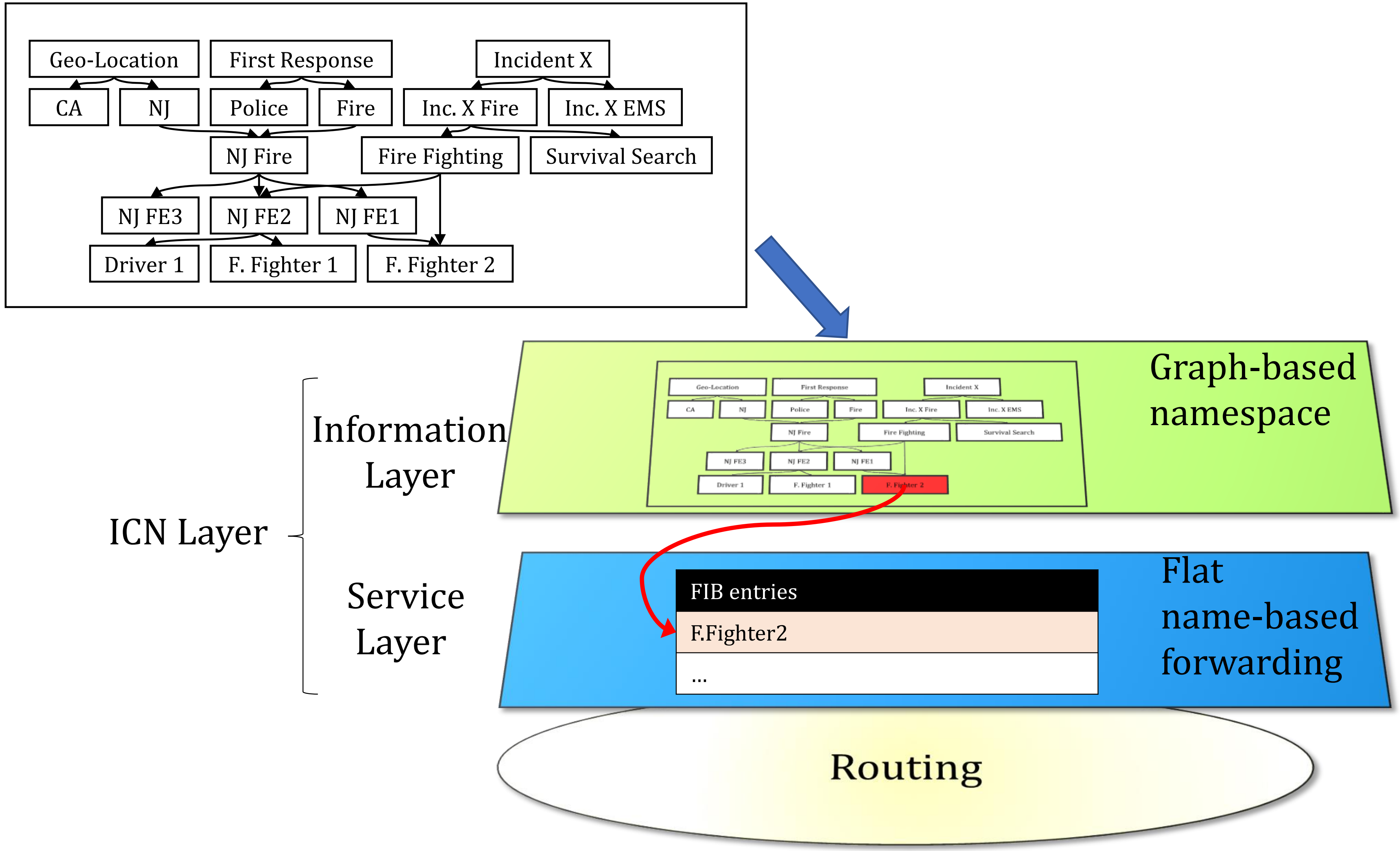
Adding a child to "F. Fighter 2" requires three modifications

To publish to "F. Fighter 2", three publications need to be made

Support graph-based namespaces in the network

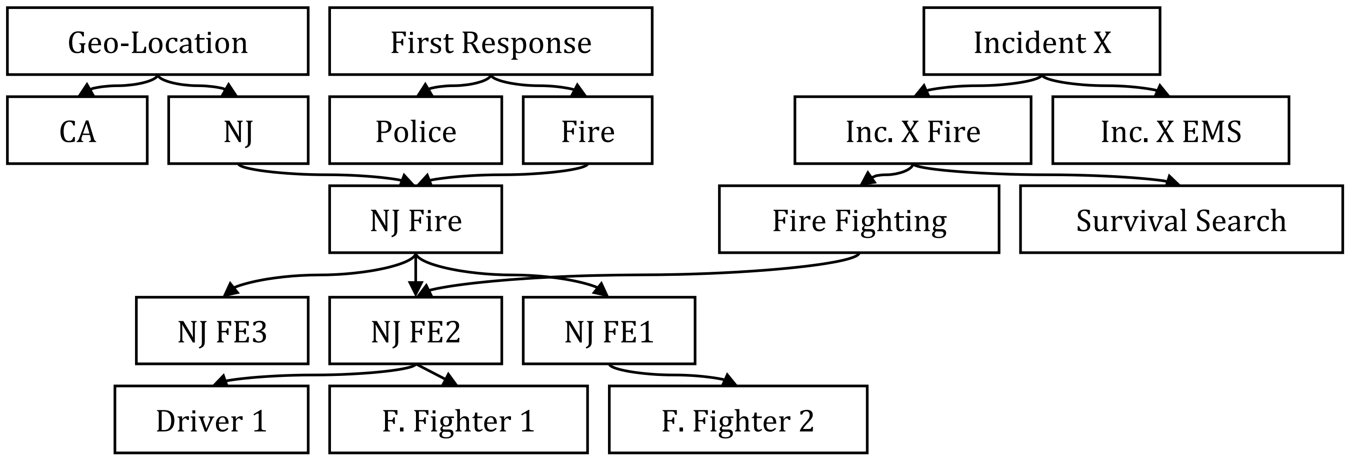
- Need support in network (multicast) for efficient delivery
- IP multicast is feasible but has issues
  - Flat IP address space, cannot capture multicast-group inter-relationship
- Information-Centric Networking (ICN) enables name-based multicast
  - However, state-of-the-art supports hierarchical naming in the network: Named Data Networking (NDN)
    - Will have to convert complex namespace graph to its hierarchical equivalent first
    - Issues: too many duplications, large FIB sizes, not very flexible with frequent namespace churning
- POISE: decouple ICN layer to Information Layer (namespace management) and Service Layer (name-based forwarding)

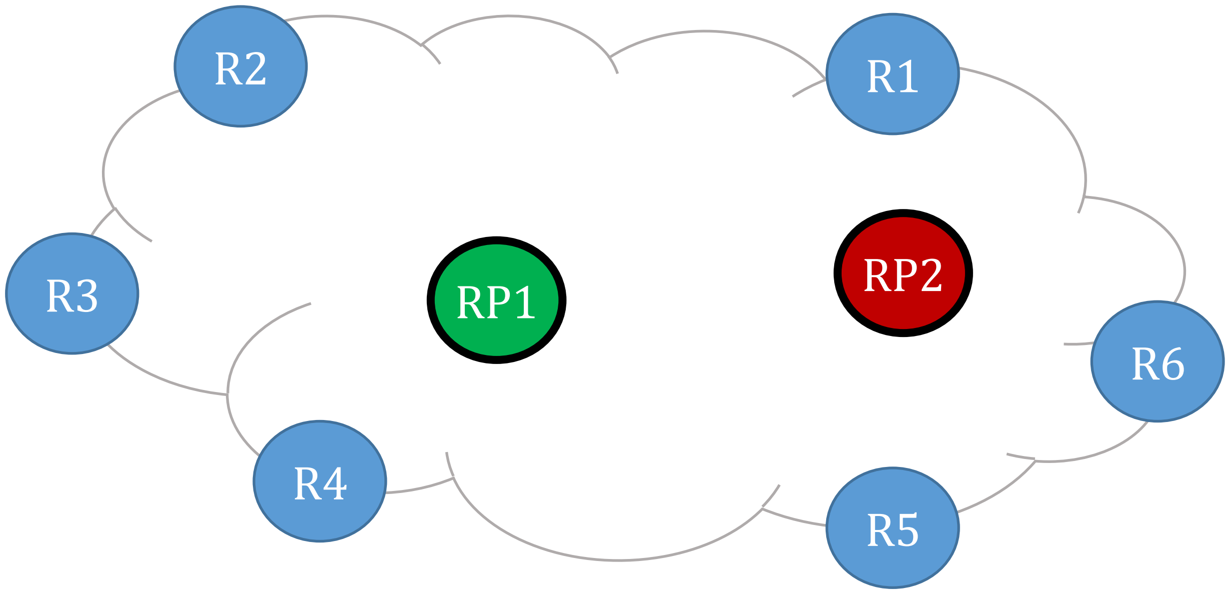
Graph-based pub/sub using POISE



Information dissemination procedure

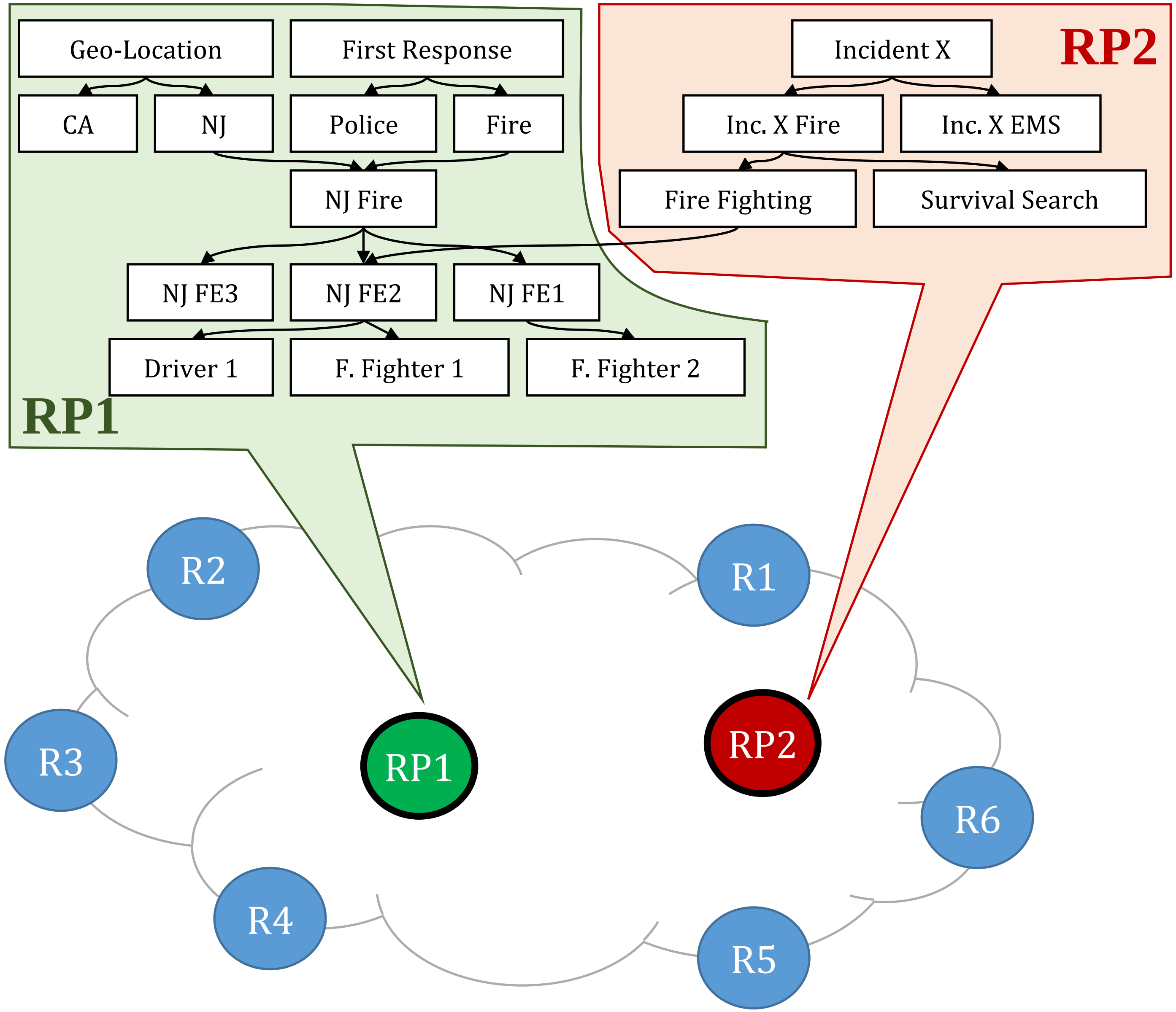
- Rendezvous Points (RPs) are distribution nodes for parts of the namespace





Information dissemination procedure

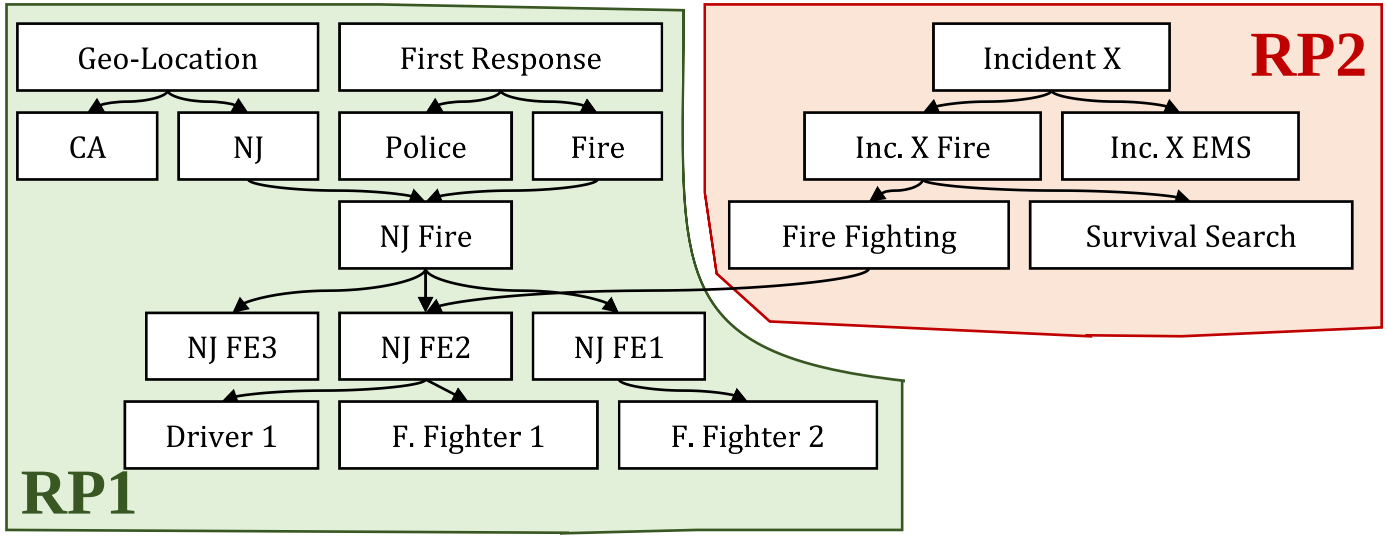
- RP1 and RP2 each maintain a (disjoint) subset of the namespace
- Name-RP mapping resolves names to RP id
  - Similar to group-to-RP mapping typical in multicast



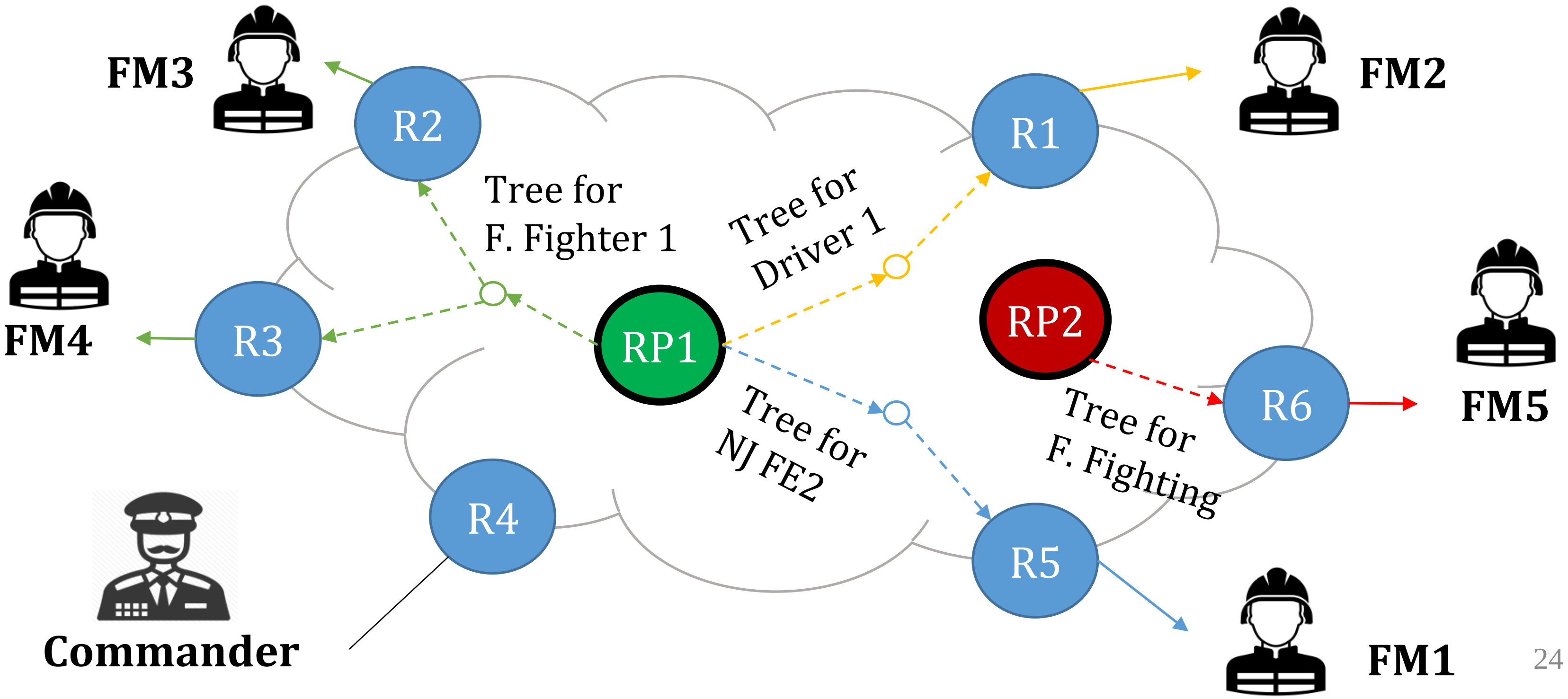
NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
...	...

Information dissemination procedure

- RPs also act as the core of multicast trees for their names
- Subscribers (firemen 1-5) join the multicast trees

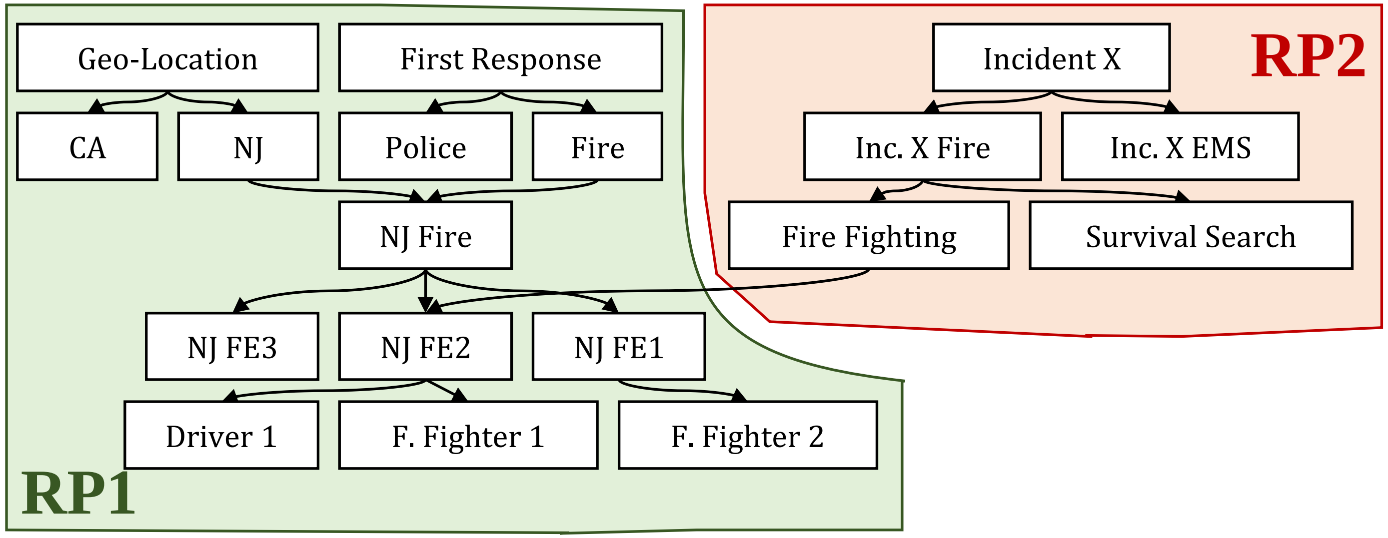


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
...	...

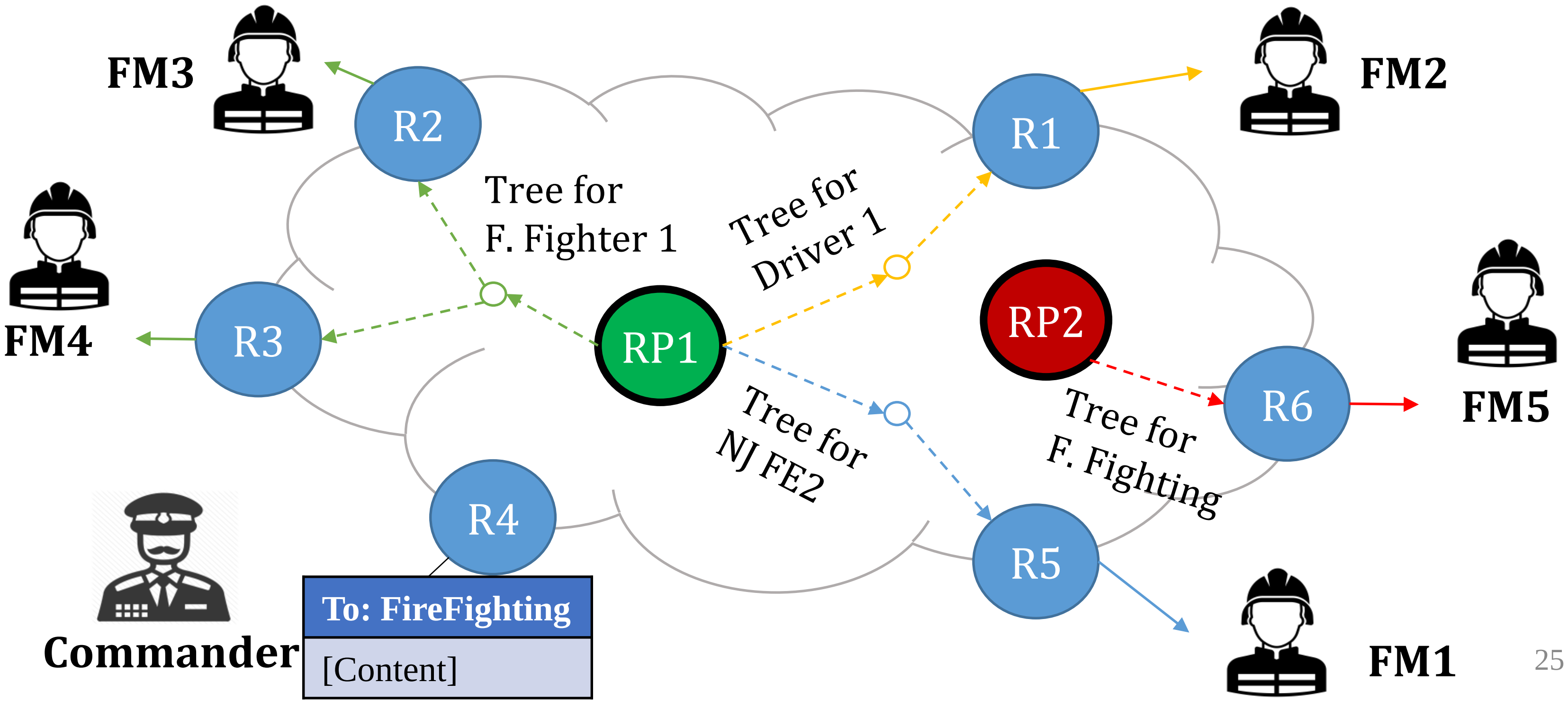


Information dissemination procedure

- Incident Commander wants to publish content (e.g., instructions) to “Fire Fighting”

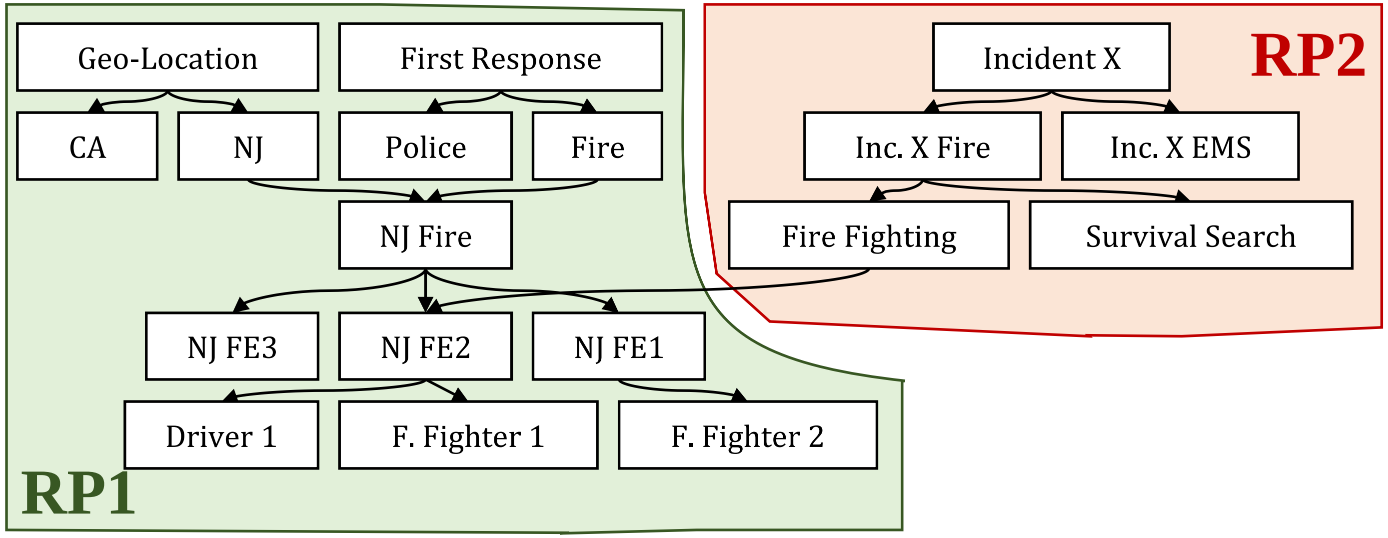


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
…	…

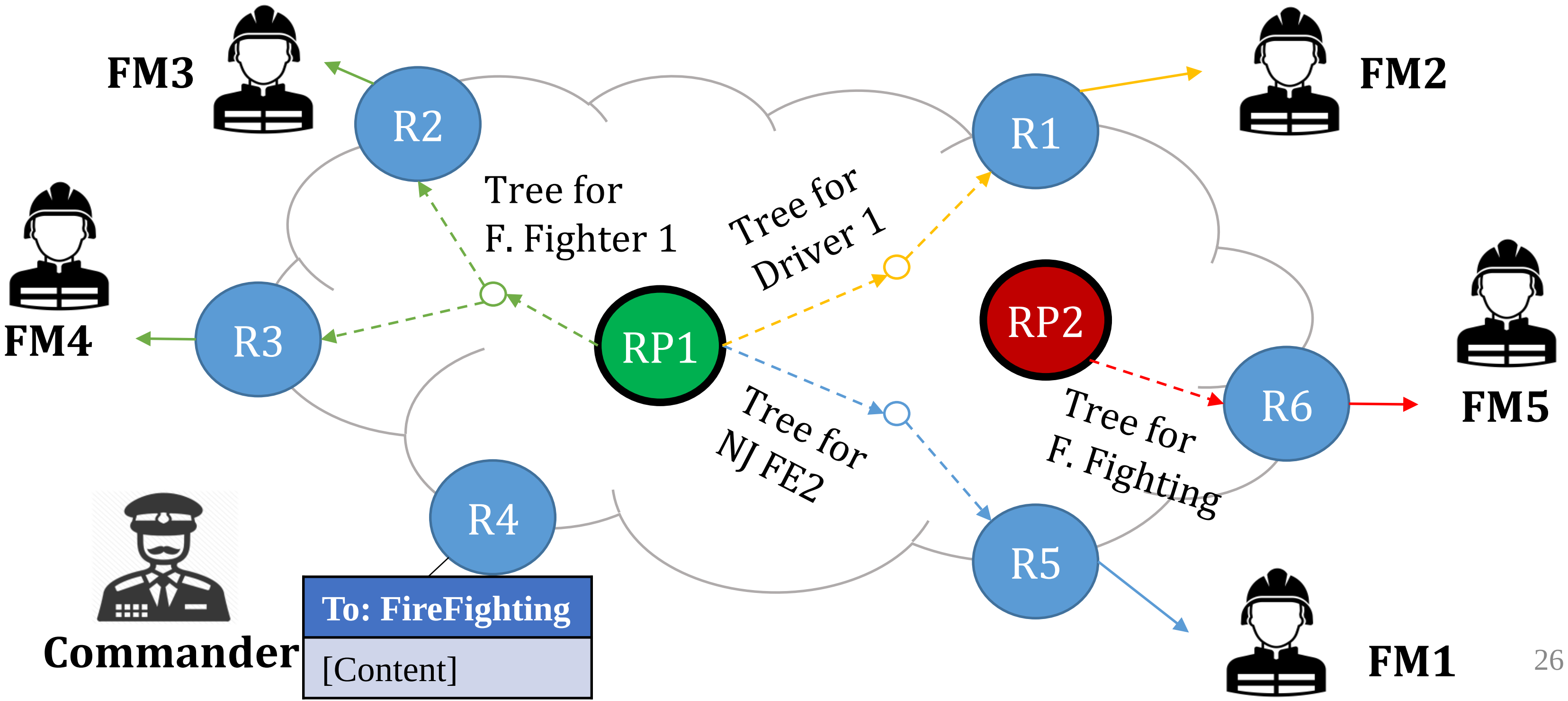


Information dissemination procedure

- Incident Commander wants to publish content (e.g., instructions) to “Fire Fighting”
  - Resolved to RP2 (look up by first-hop router R4)

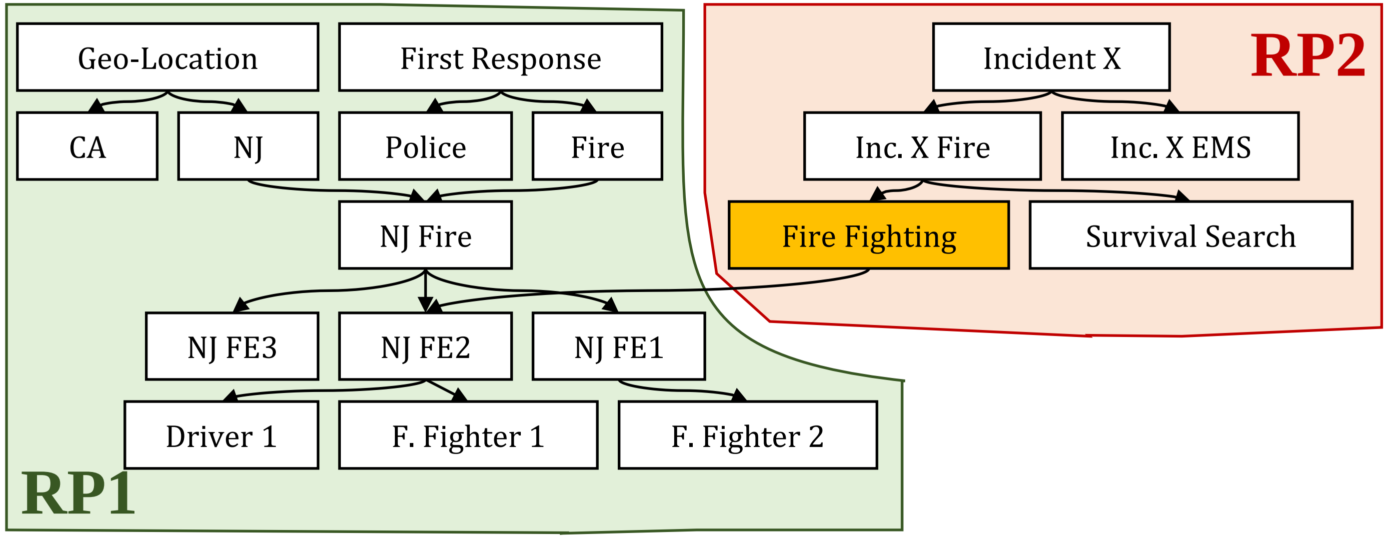


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
…	…

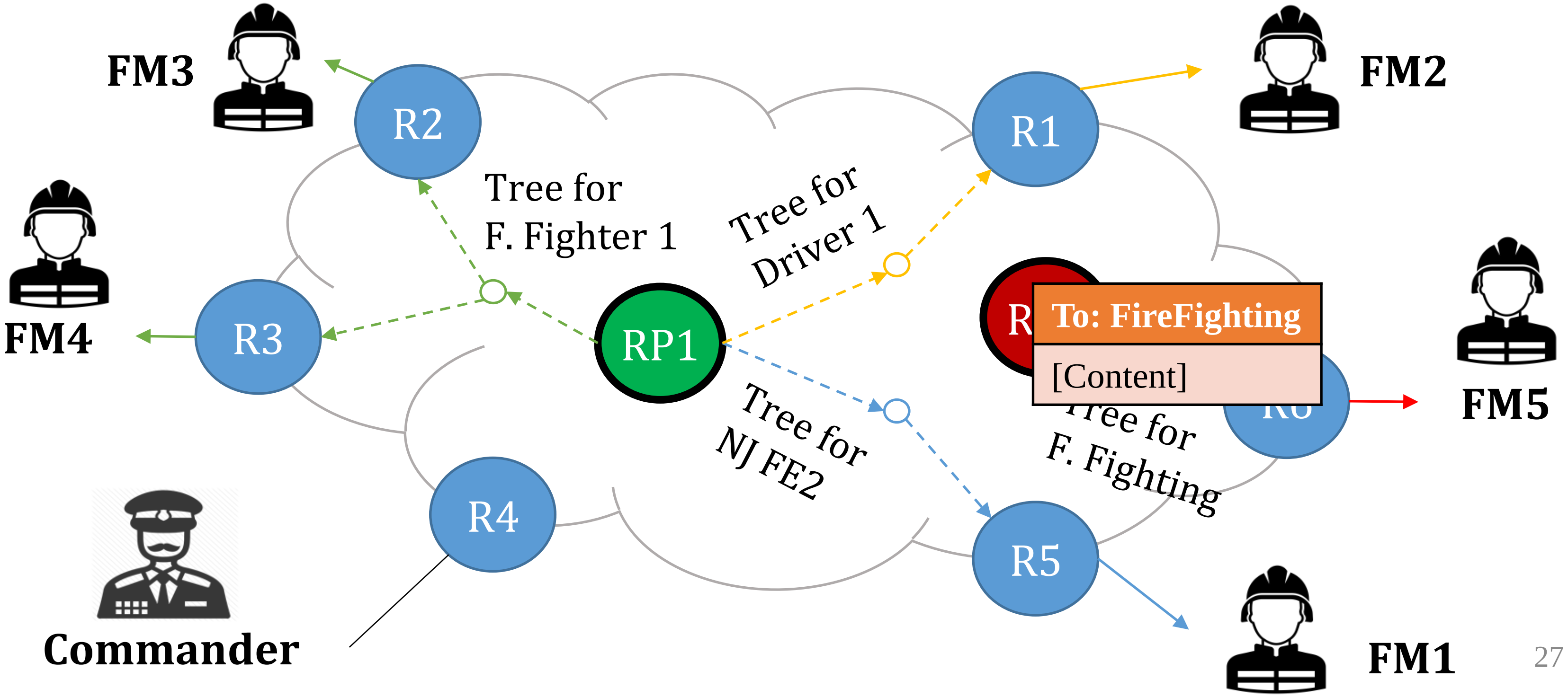


Information dissemination procedure

- At RP
  - Multicast to name and descendants on the same RP

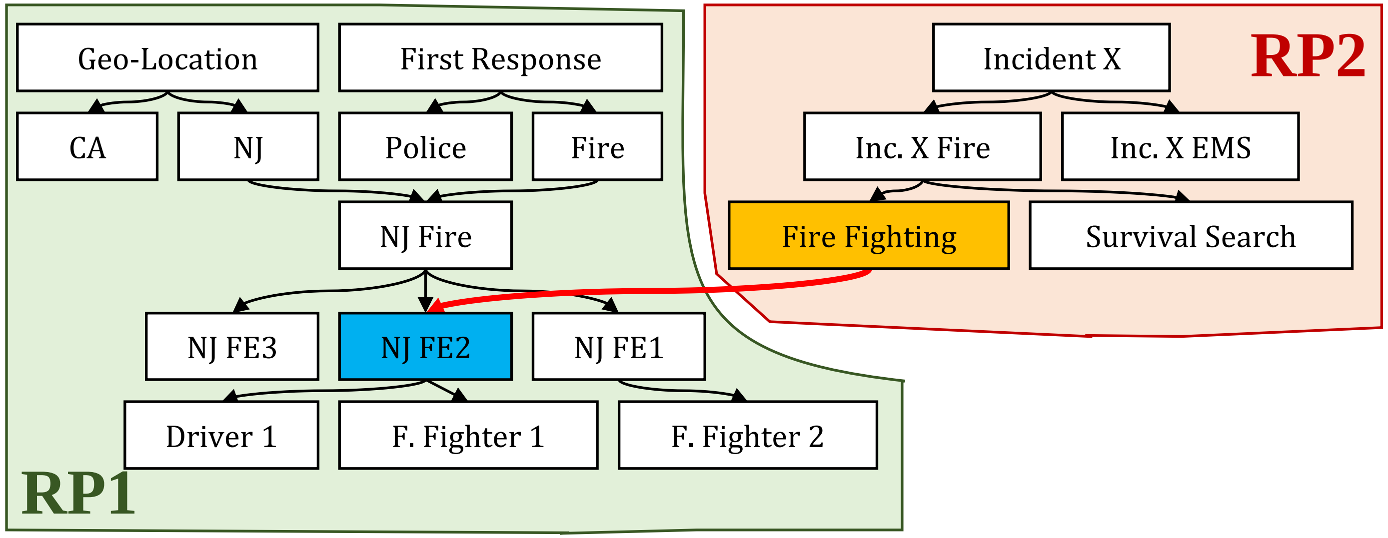


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
...	...

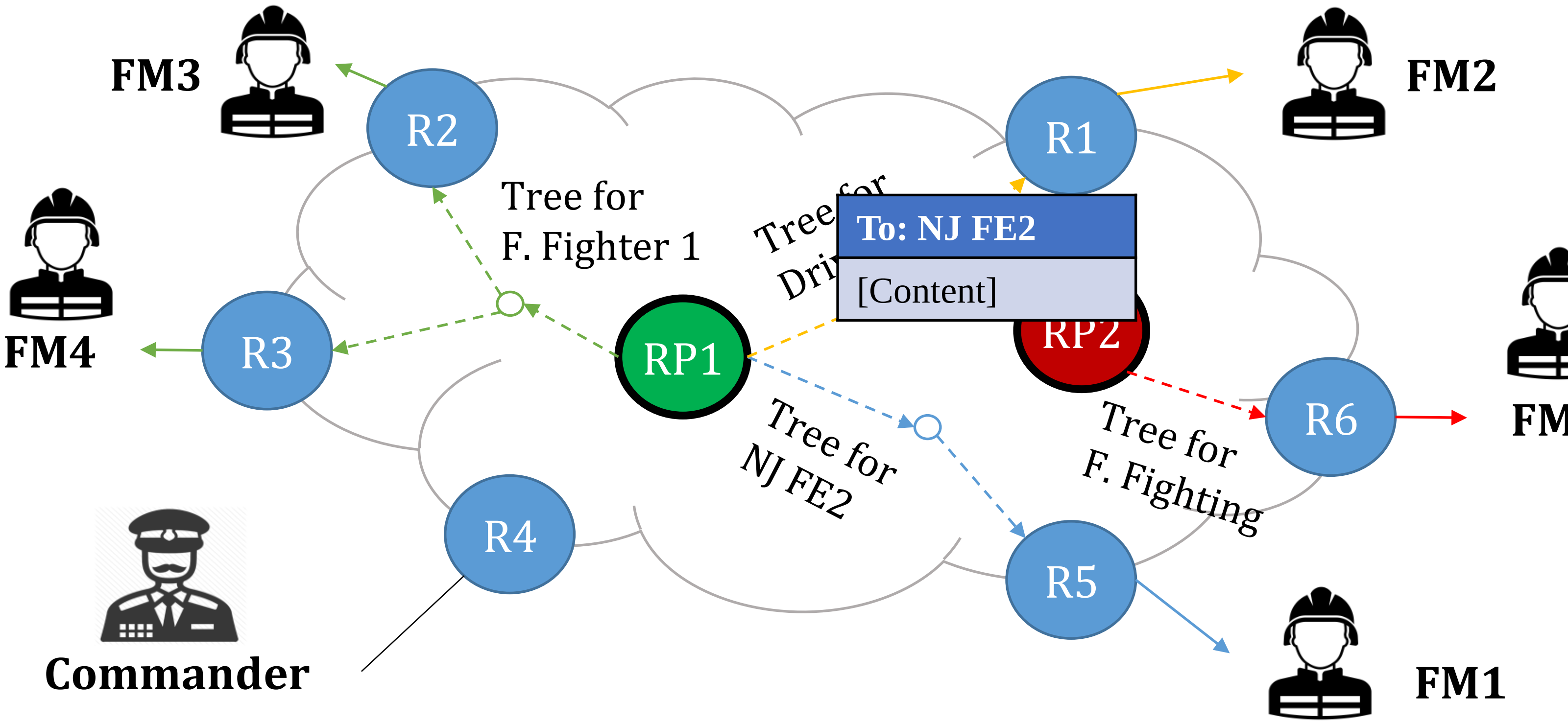


Information dissemination procedure

- At RP
  - Multicast to name and descendants on the same RP
  - Unicast to name if on another RP

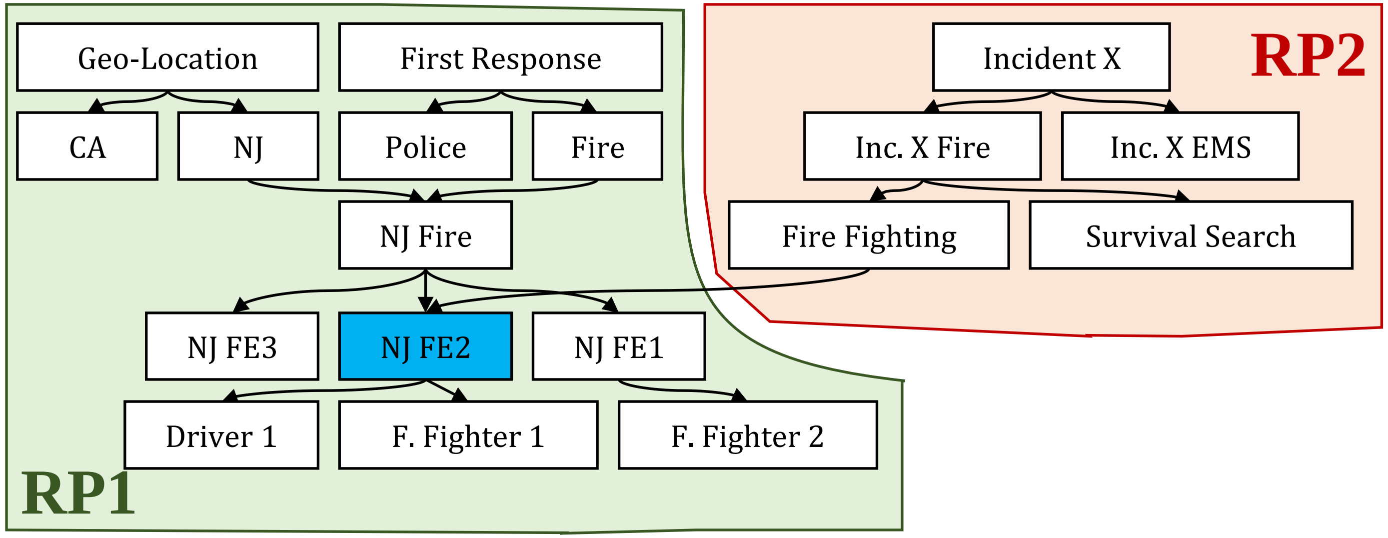


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
…	…

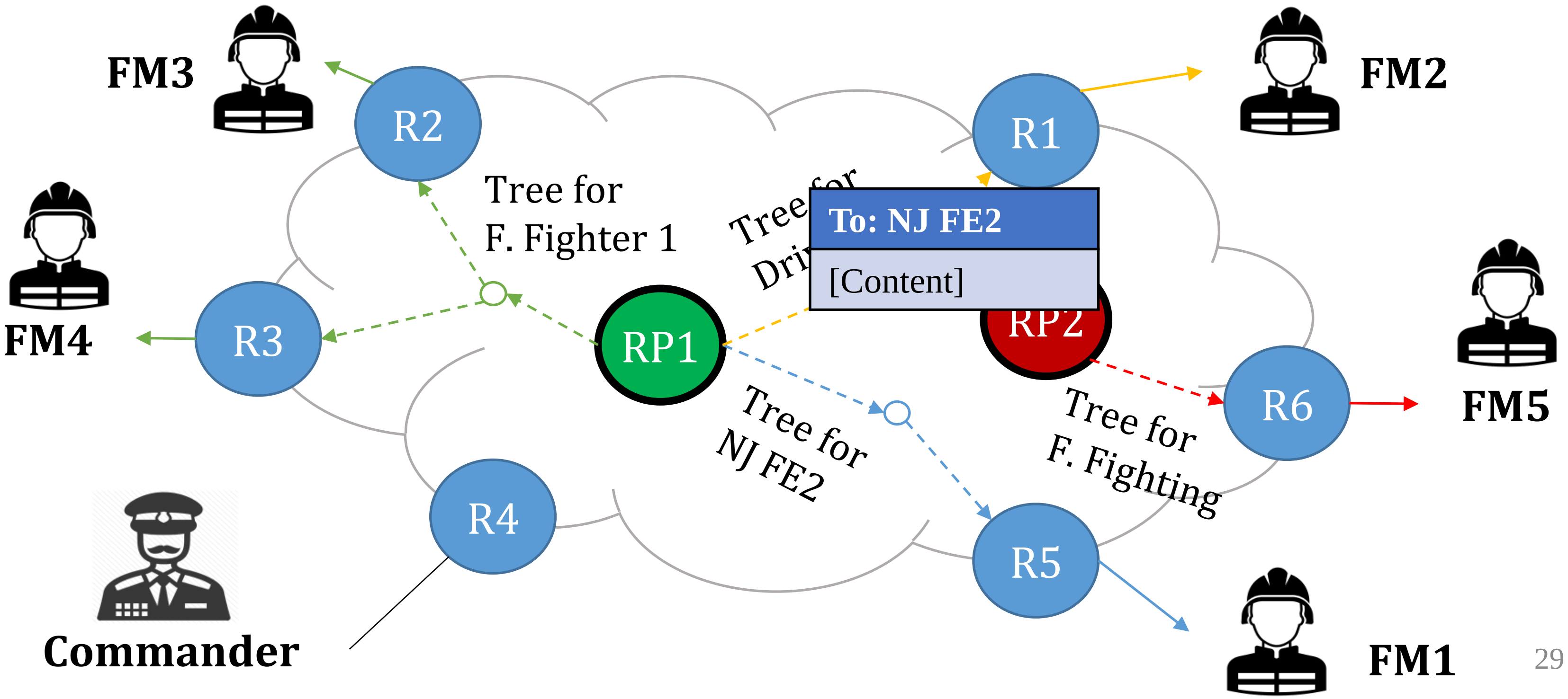


Information dissemination procedure

- At RP
  - Multicast to name and descendants on the same RP
  - Unicast to name if on another RP

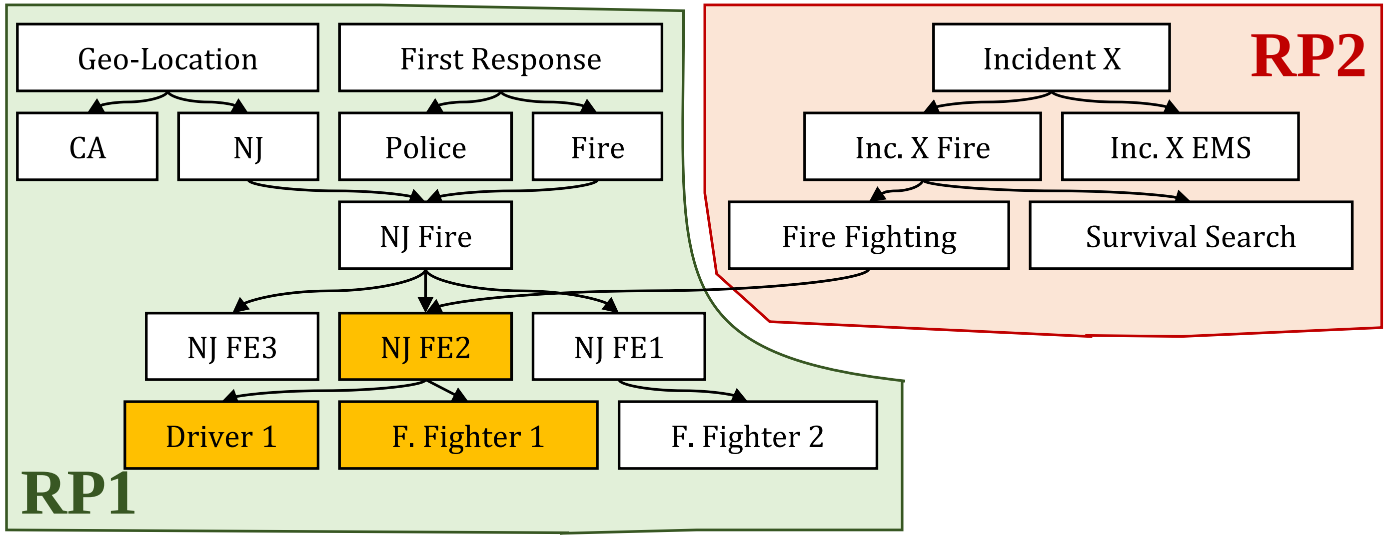


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
…	…

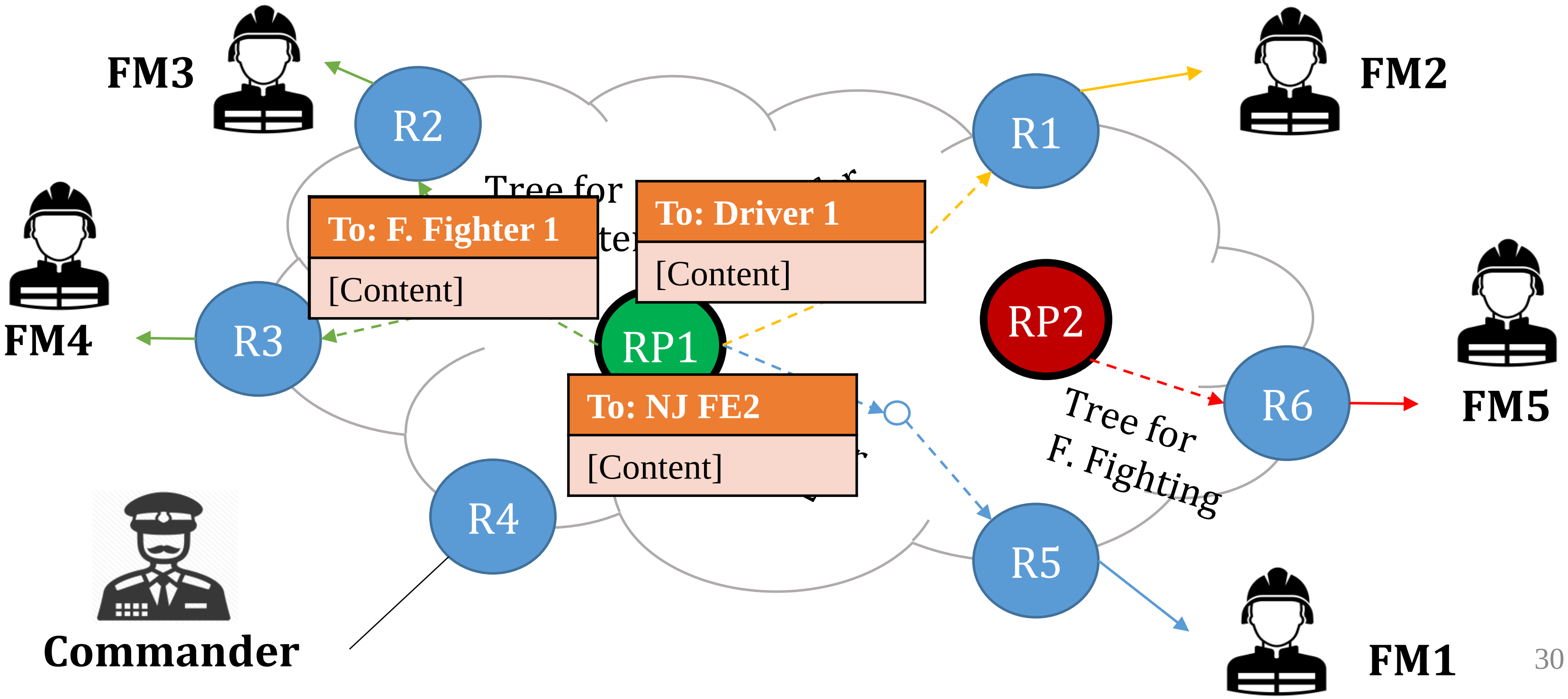


Information dissemination procedure

- At RP
  - Multicast to name and descendants on the same RP
  - Unicast to name if on another RP
- All subscribers of "Fire Fighting" and all its descendants receive the publication

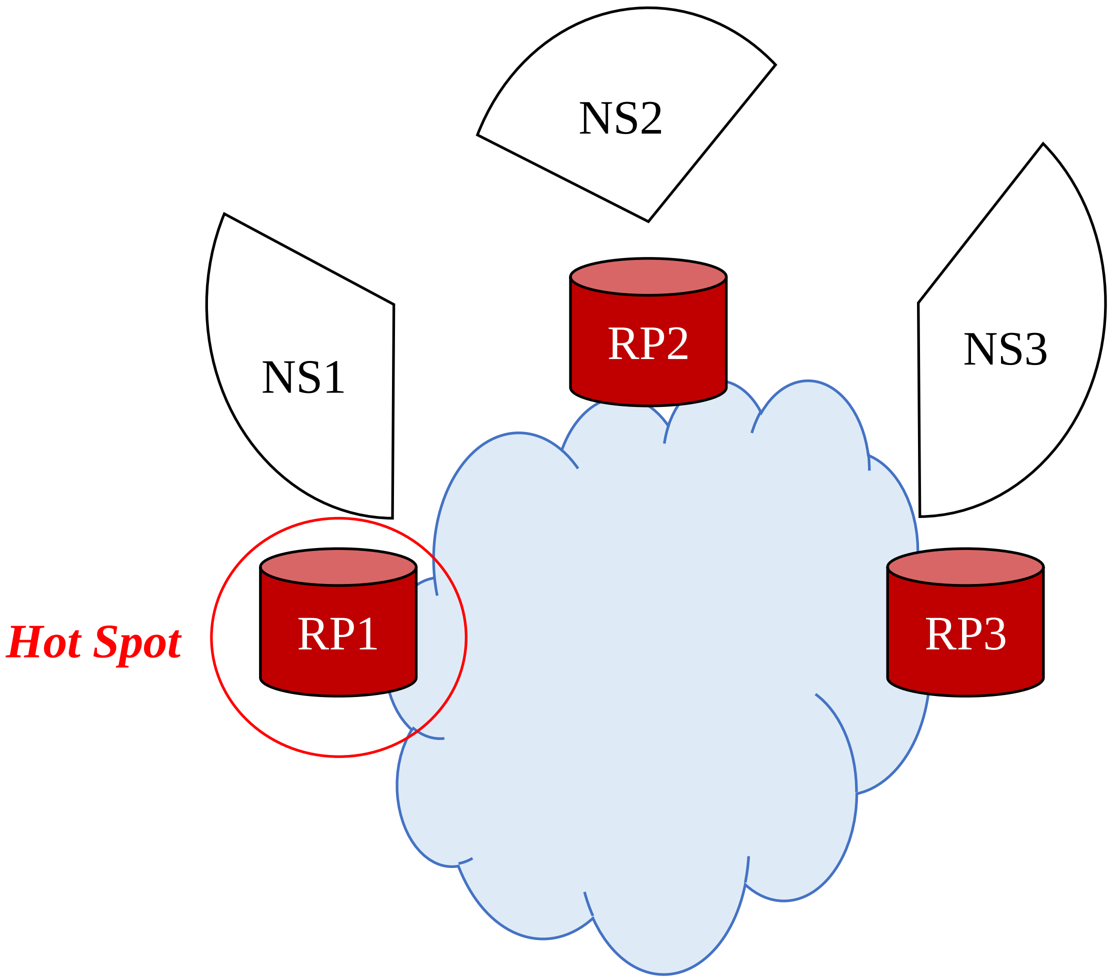


NAME-RP Mapping	
NAME	**RP**
Geo-Location	RP1
NJ FE2	RP1
Driver 1	RP1
F. Fighter 1	RP1
Incident X	RP2
Fire Fighting	RP2
…	…



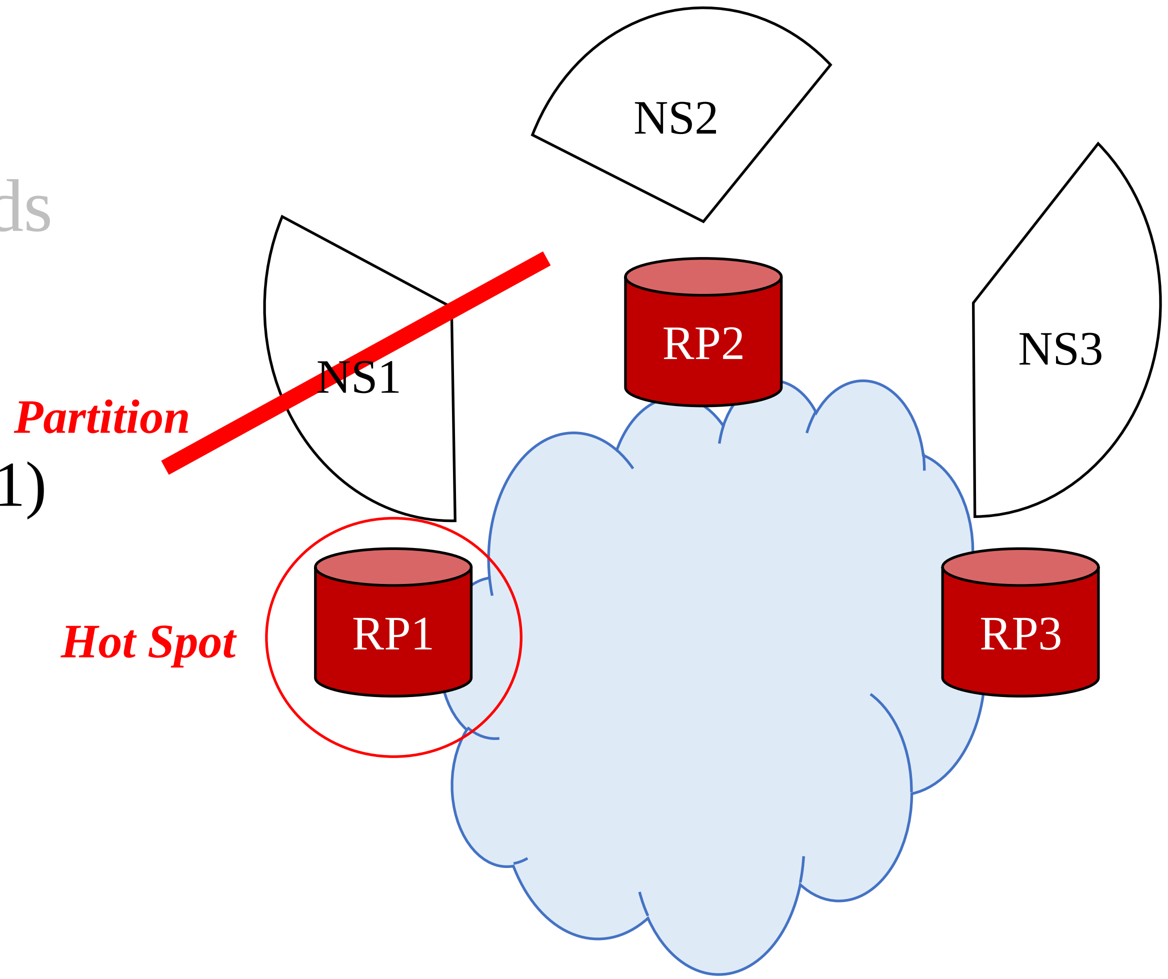
Load splitting

- Different RPs experience different workloads
  - One RP may become a “hot spot” (RP1)



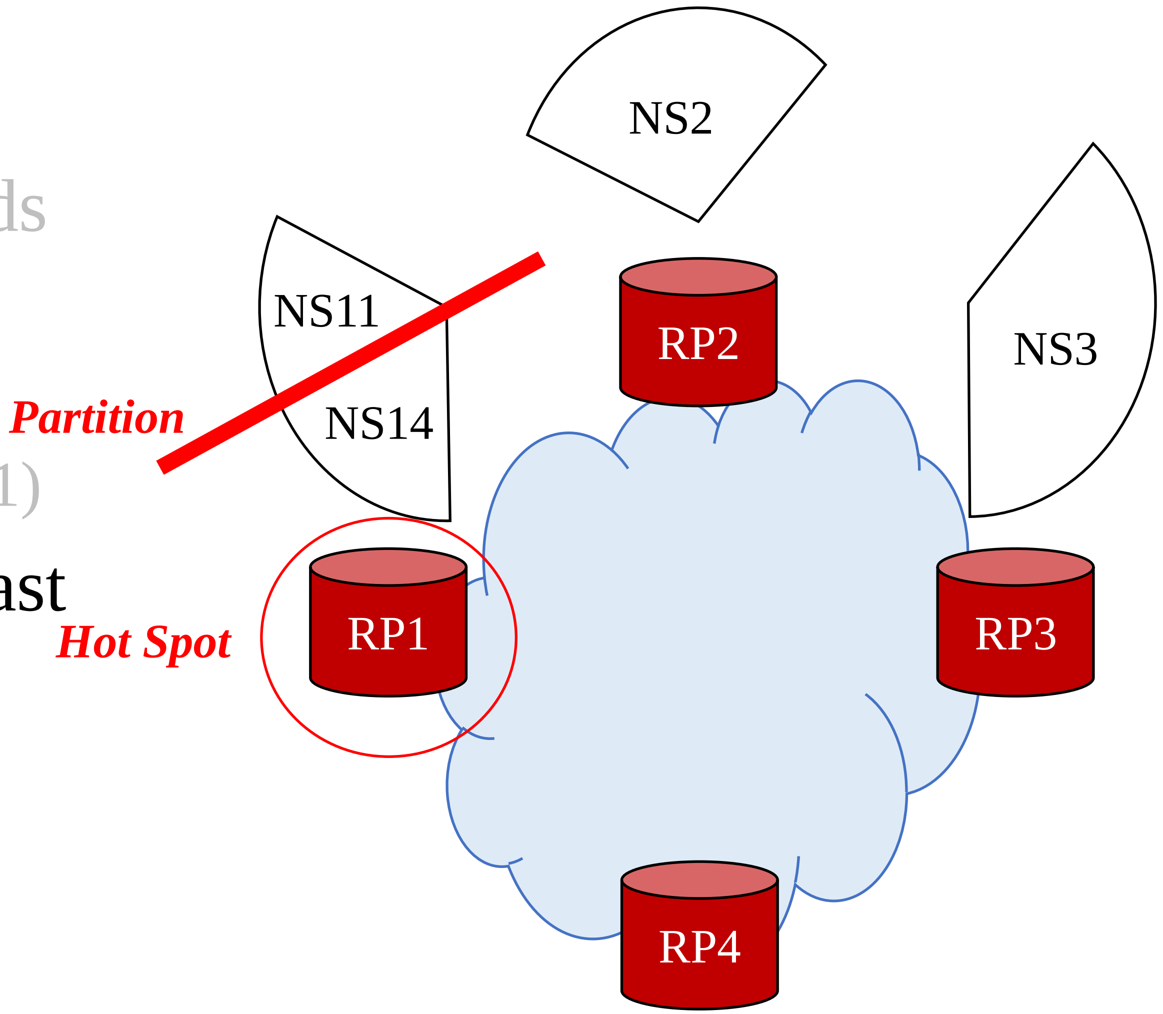
Load splitting

- Different RPs experience different workloads
  - One RP may become a “hot spot” (RP1)
- To eliminate this traffic concentration
  - Partition its local namespace graph (NS1 at RP1)



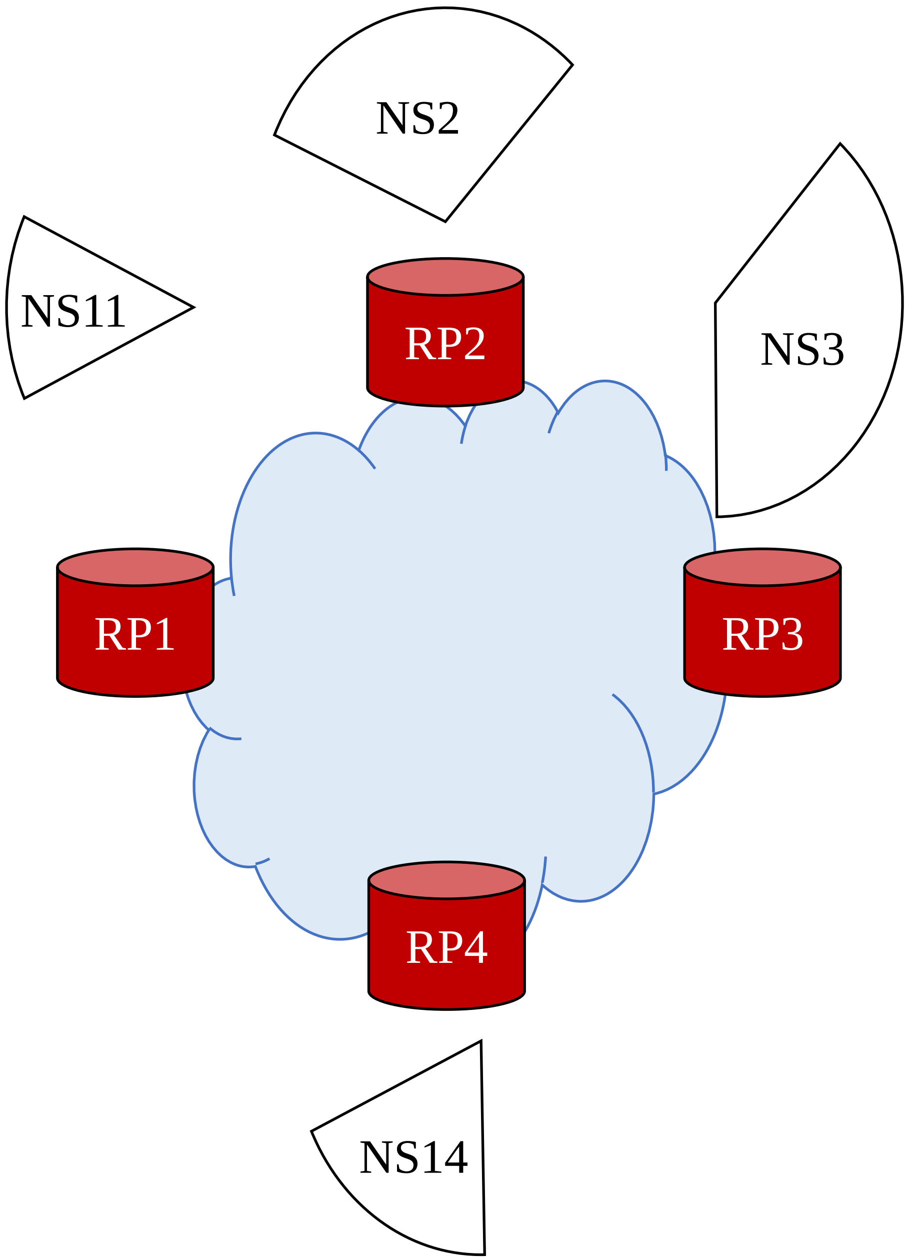
Load splitting

- Different RPs experience different workloads
  - One RP may become a "hot spot" (RP1)
- To eliminate this traffic concentration
  - Partition its local namespace graph (NS1 at RP1)
- Migrate one segment (NS14) and its multicast trees to a new RP (RP4)
- Now RP1 is not a hot spot anymore

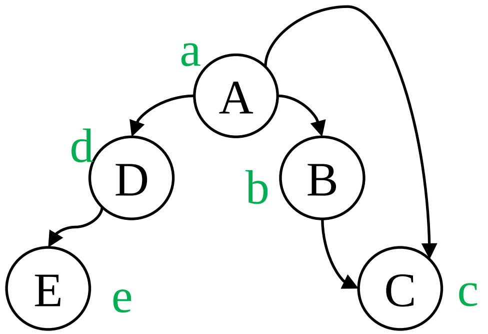


Load splitting

- Different RPs experience different workloads
  - One RP may become a “hot spot” (RP1)
- To eliminate this traffic concentration
  - Partition its local namespace graph (NS1 at RP1)
- Migrate one segment (NS14) and its multicast trees to a new RP (RP4)
- Now RP1 is not a hot spot anymore
- POISE provides
  - A workload-driven graph partitioning algorithm to find a balanced partitioning
  - A seamless, reliable namespace migration



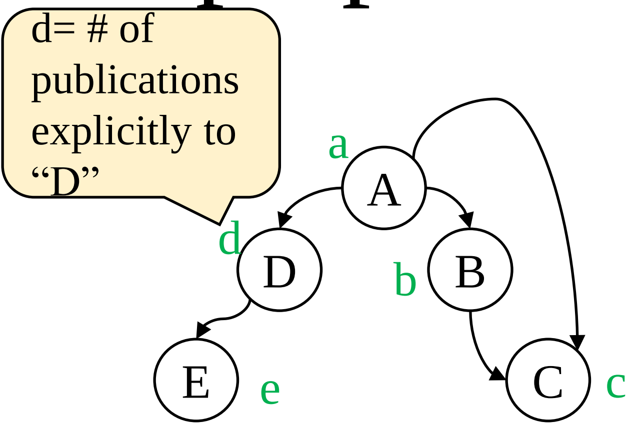
Graph partitioning



Initial Graph

- Workload at RP represented as a labeled directed namespace graph
  - Nodes (names) initially labeled with explicit incoming request count in recent time window

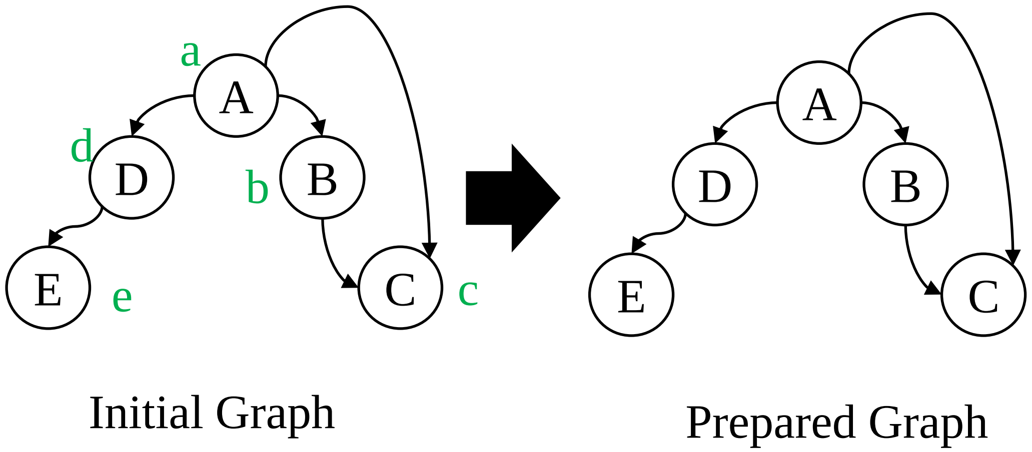
Graph partitioning



Initial Graph

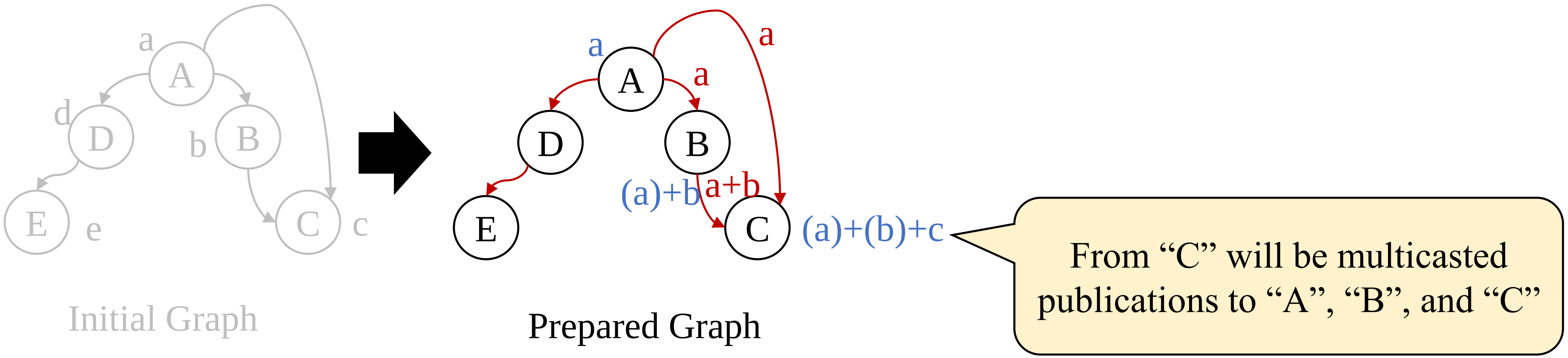
- Workload at RP represented as a labeled directed namespace graph
  - Nodes (names) initially labeled with explicit incoming request count in recent time window
    - Example: "d" publications sent to name "D"
  - Goal: find a "good" partitioning to cut the namespace graph to two segments

Graph partitioning



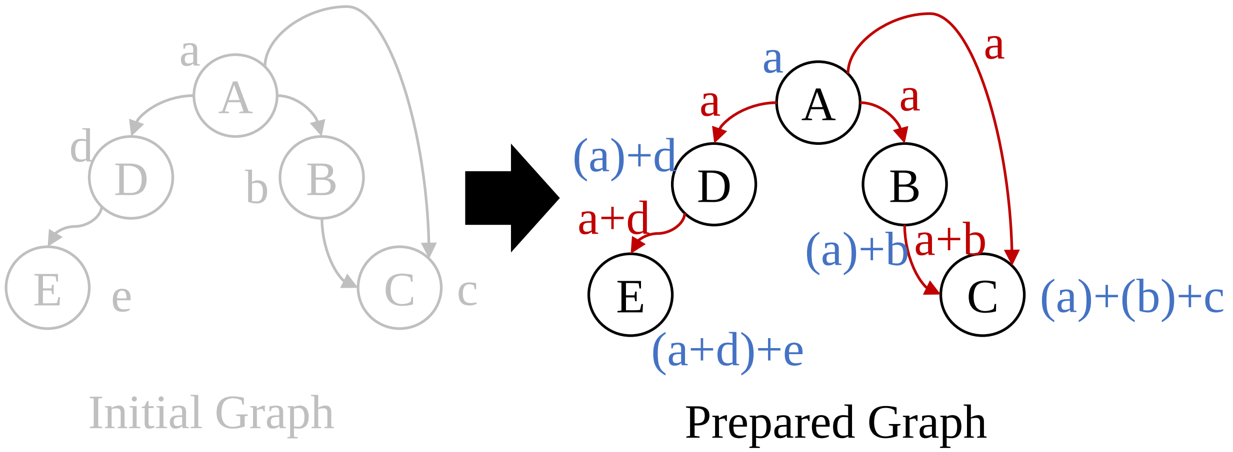
- Prepared (partitionable) graph with multicast workloads added

Graph partitioning



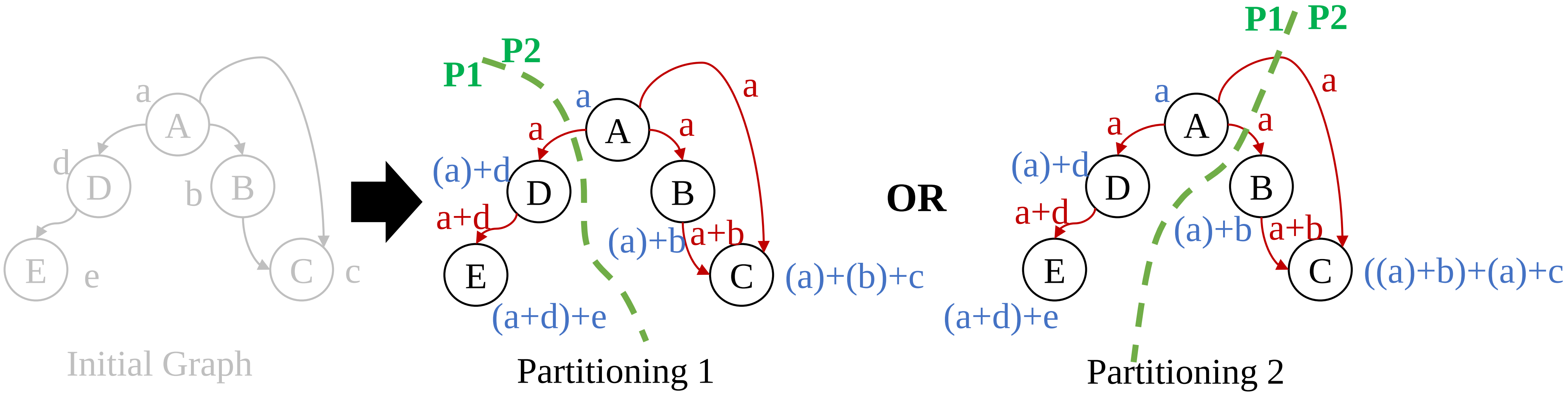
- Prepared (partitionable) graph with multicast workloads added
  - Node weights = # of messages to be multicasted for subscribers of node (e.g.: C)
    - Includes explicit publications to "C" plus publications to ancestors of "C"
  - Edge weights = # of messages going towards the child nodes

Graph partitioning



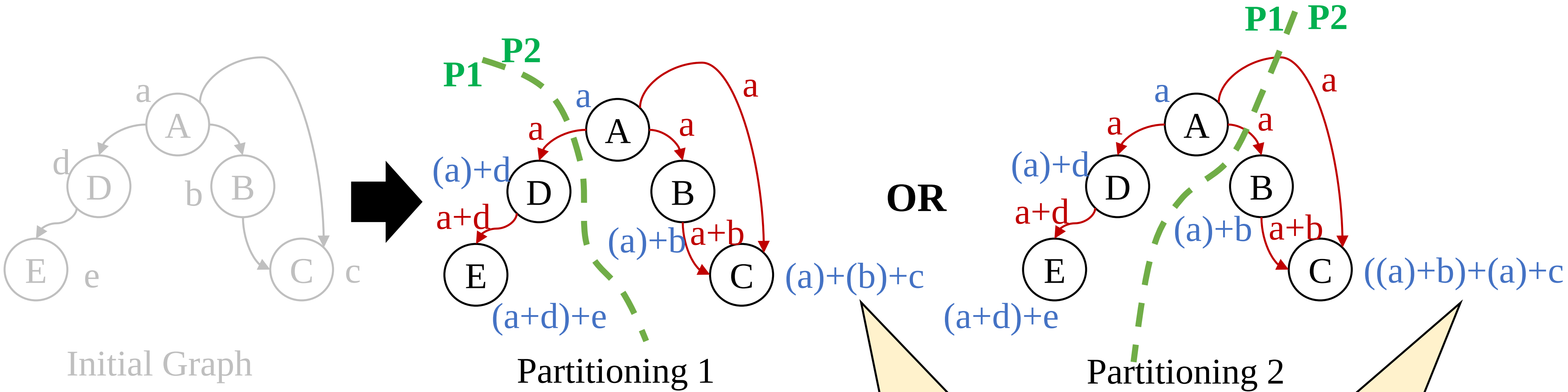
- Prepared (partitionable) graph with multicast workloads added
  - Node weights = # of messages to be multicasted for subscribers of node (e.g.: C)
    - Includes explicit publications to “C” plus publications to ancestors of “C”
  - Edge weights = # of messages going towards the child nodes

Graph partitioning



- Many ways to partition the graph

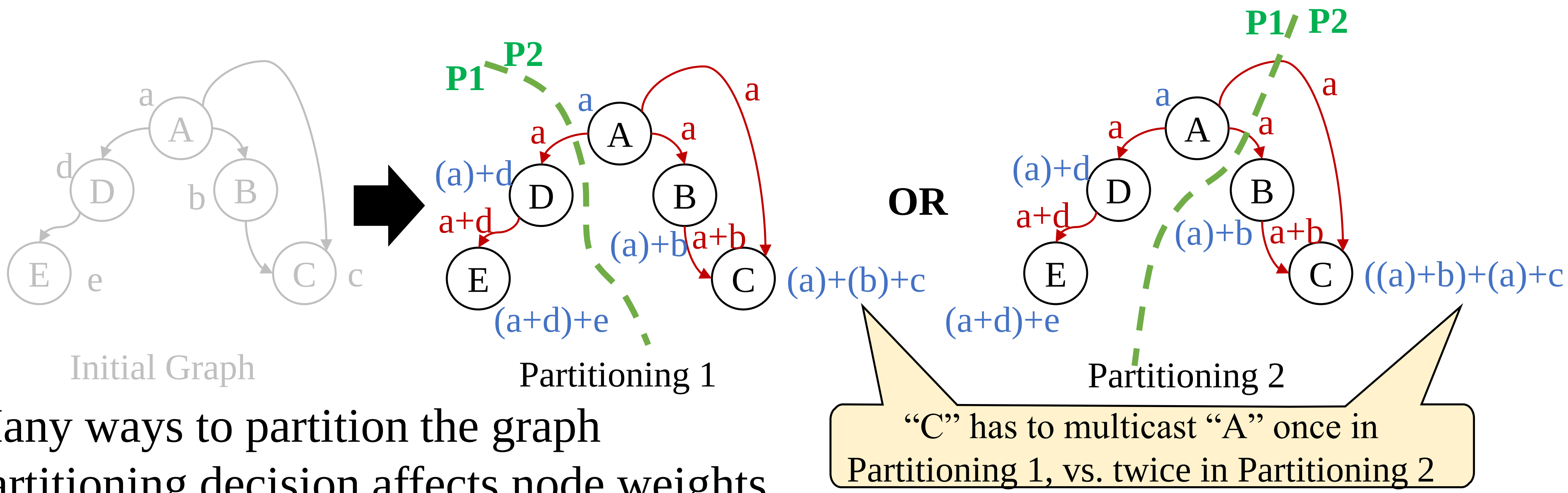
Graph partitioning



"C" has to multicast "A" once in Partitioning 1, vs. twice in Partitioning 2

- Many ways to partition the graph
- Partitioning decision affects node weights
  - In Partitioning 1, weight of "C" is counting input from "A" once, twice in Partitioning 2
    - Two paths from A to C: both contained in one segment vs. both going across the cut

Graph partitioning



- Many ways to partition the graph
- Partitioning decision affects node weights
  - In Partitioning 1, weight of “C” is counting input from “A” once, twice in Partitioning 2
    - Two paths from A to C: both contained in one segment vs. both going across the cut
- “Chicken and egg problem”
  - Objective function is a complex function of partitioning itself → Complex Objectives
  - State-of-the-art graph partitioners, such as METIS, fall short
    - METIS: Graph partitioner, high quality and fast; “*gold standard in partitioning*”
- **POISE: hybrid graph partitioning: heuristic (METIS) + meta-heuristic (Tabu Search)**

Graph partitioning

1. Prepare weighted graph (diffusion method)
2. Provide initial solution using METIS
3. Tabu search and report best solution found before stop
   - Objective: Minimize weighted function $F(G1, G2)$ for two segments $G1$ and $G2$
   - $F(G1, G2) = \alpha . |TC(G1) - TC(G2)| + \beta . \max\left(TC(G1), TC(G2)\right) + \gamma . (UC(G1) + UC(G2))$

Graph partitioning

1. Prepare weighted graph (diffusion method)
2. Provide initial solution using METIS
3. Tabu search and report best solution found before stop
    - Objective: Minimize weighted function $F(G1, G2)$ for two segments $G1$ and $G2$
    - $F(G1, G2) = \alpha . |TC(G1) - TC(G2)| + \beta . \max\big(TC(G1), TC(G2)\big) + \gamma . (UC(G1) + UC(G2))$
        - Minimize imbalance of total workload (#total messages)

Graph partitioning

1. Prepare weighted graph (diffusion method)
2. Provide initial solution using METIS
3. Tabu search and report best solution found before stop
   - Objective: Minimize weighted function $F(G1, G2)$ for two segments $G1$ and $G2$
   - $F(G1, G2) = \alpha . |TC(G1) - TC(G2)| + \beta . \max(TC(G1), TC(G2)) + \gamma . (UC(G1) + UC(G2))$
     - Minimize imbalance of total workload (#total messages)
     - Minimize the maximum total workload of either segment

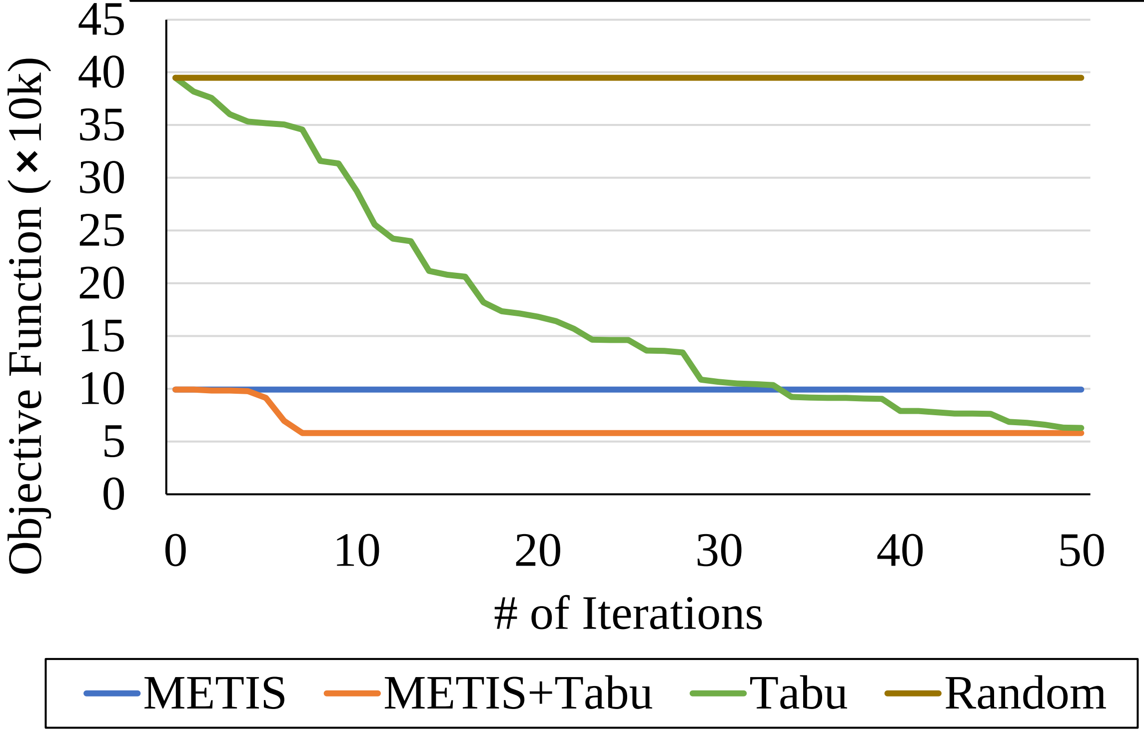
Graph partitioning

1. Prepare weighted graph (diffusion method)
2. Provide initial solution using METIS
3. Tabu search and report best solution found before stop
    - Objective: Minimize weighted function $F(G1, G2)$ for two segments $G1$ and $G2$
    - $F(G1, G2) = \alpha . |TC(G1) - TC(G2)| + \beta . \max\left(TC(G1), TC(G2)\right) + \gamma . (UC(G1) + UC(G2))$
        - Minimize imbalance of total workload (#total messages)
        - Minimize the maximum total workload of either segment
        - Minimize total unicast workload (inter-RP communication)

Objective: $F(G1, G2) = \alpha.|TC(G1) - TC(G2)| + \beta.\max(TC(G1), TC(G2)) + \gamma.(UC(G1) + UC(G2))$

Graph partitioning

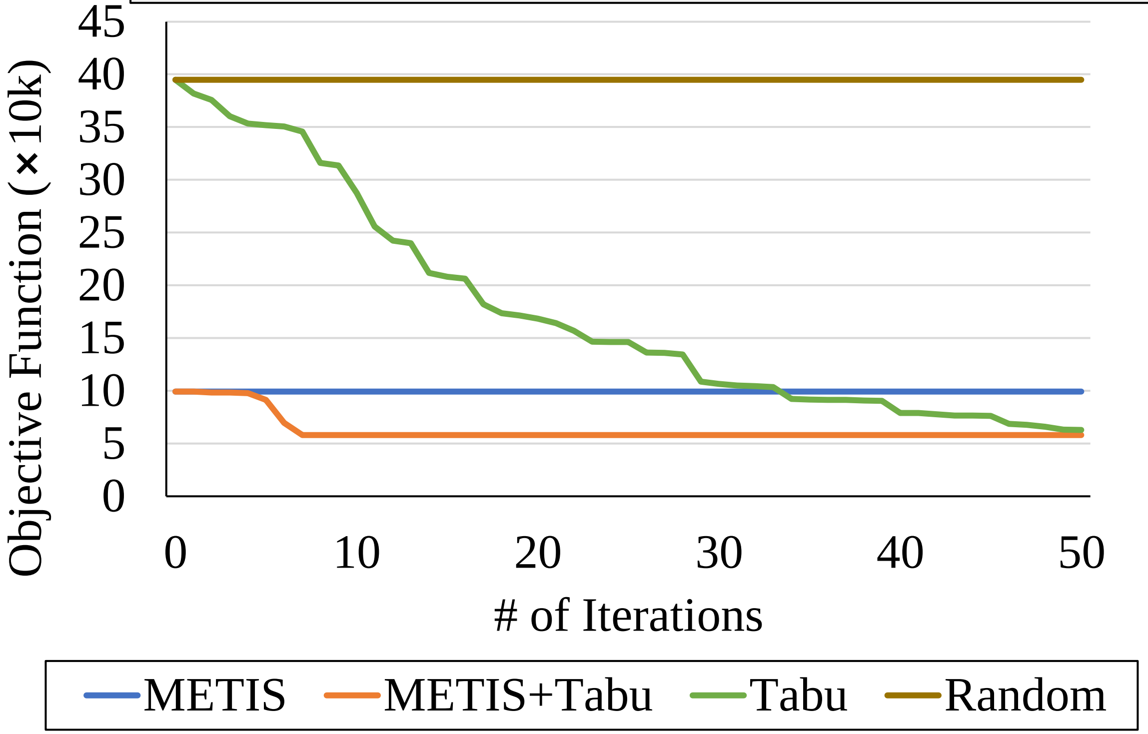
- POISE: METIS+Tabu outperforms other choices, on a graph G(50,84) *
  - METIS-only
  - Tabu-only
  - Random
- Impact of # of refinement iterations on quality of solution



* *Input graphs from repository at "www.graphdrawing.org/data.html"*

Graph partitioning

Objective: $F(G1, G2) = \alpha.|TC(G1) - TC(G2)| + \beta.\max(TC(G1), TC(G2)) + \gamma.(UC(G1) + UC(G2))$



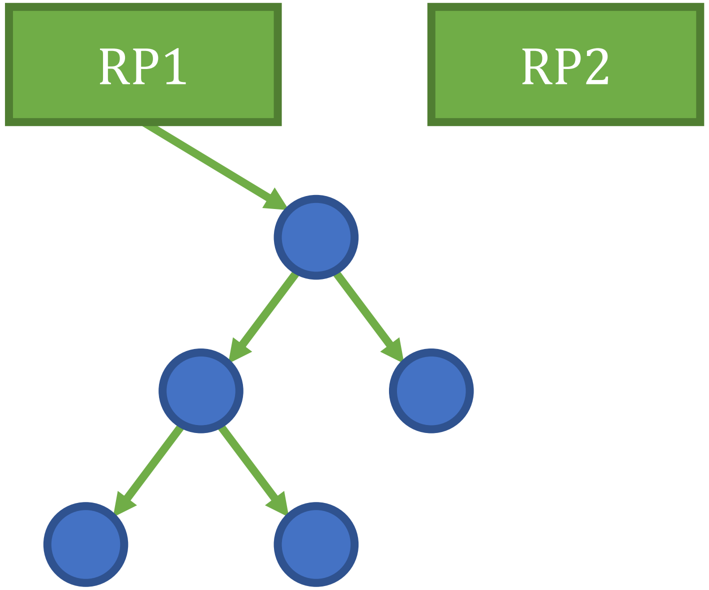
- POISE: METIS+Tabu outperforms other choices, on a graph G(50,84)
  - METIS-only
  - Tabu-only
  - Random
- Impact of # of refinement iterations on quality of solution
- Evaluate with different graphs
  - METIS+Tabu (POISE) consistently better quality than METIS

Vertices	Edges	METIS	POISE
10	14	2,093	1,916
10	18	2,988	2,319
10	28	5,170	2,873
50	75	11,159	3,820
50	84	99,292	57,897
100	191	25,858	20,470

** Input graphs from repository at "www.graphdrawing.org/data.html"*

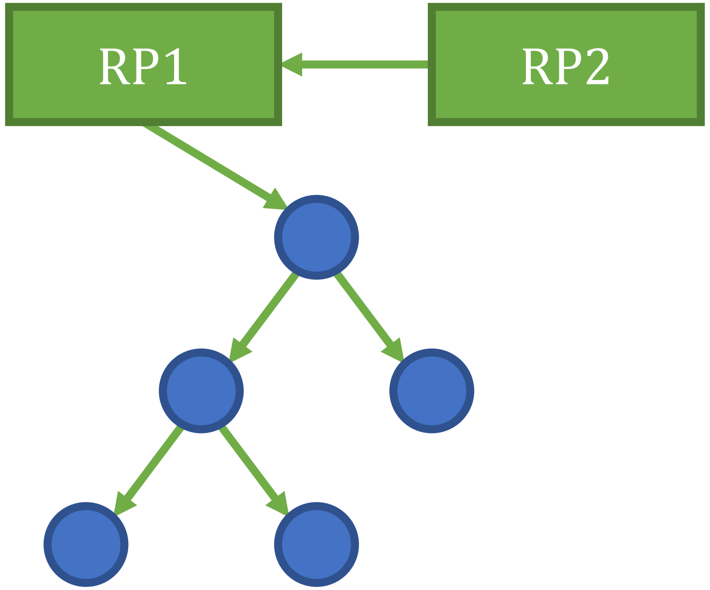
Core migrations

- Goal: seamless and reliable core migration
- Example: migrate a tree from RP1 to RP2



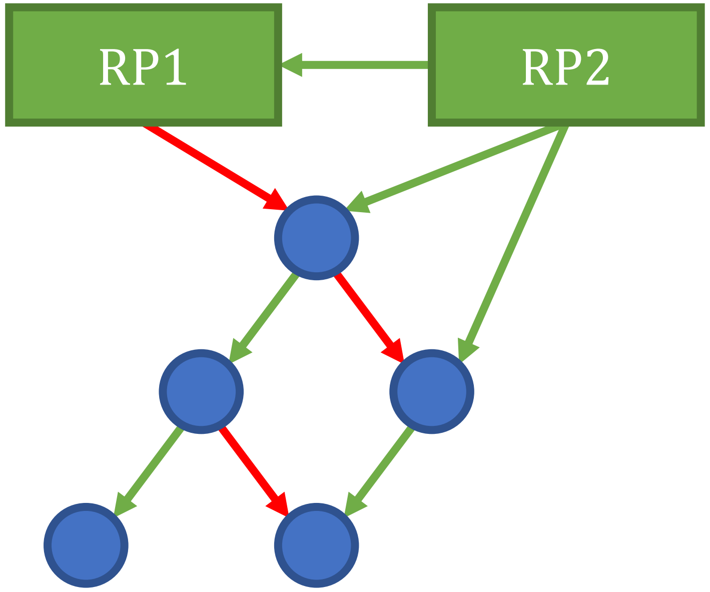
Core migrations

- Goal: seamless and reliable core migration
- Example: migrate a tree from RP1 to RP2
- RP1 notifies RP2 and subscribe to it; notifies the network to update NAME-RP mapping



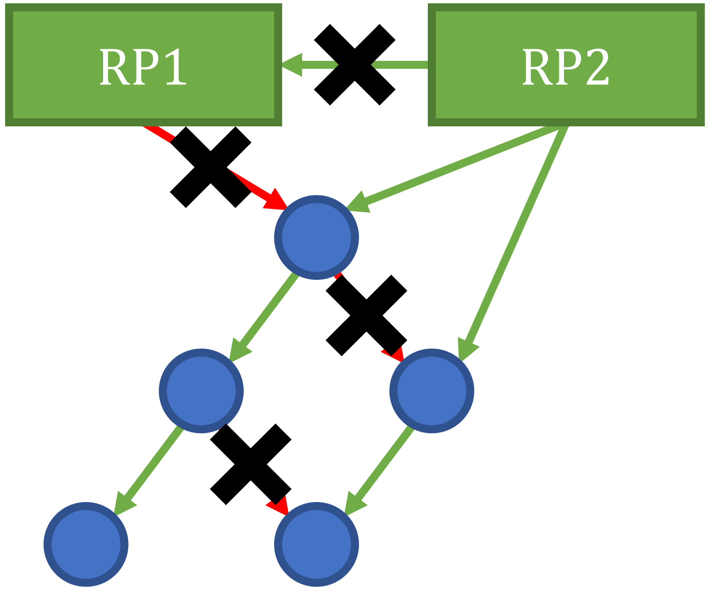
Core migrations

- Goal: seamless and reliable core migration
- Example: migrate a tree from RP1 to RP2
- RP1 notifies RP2 and subscribe to it; notifies the network to update NAME-RP mapping
- RP1 sends a marker packet M1 to all nodes in the tree
  - Nodes join the new multicast tree at RP2; keep the old paths toward RP1 to ensure reliable delivery



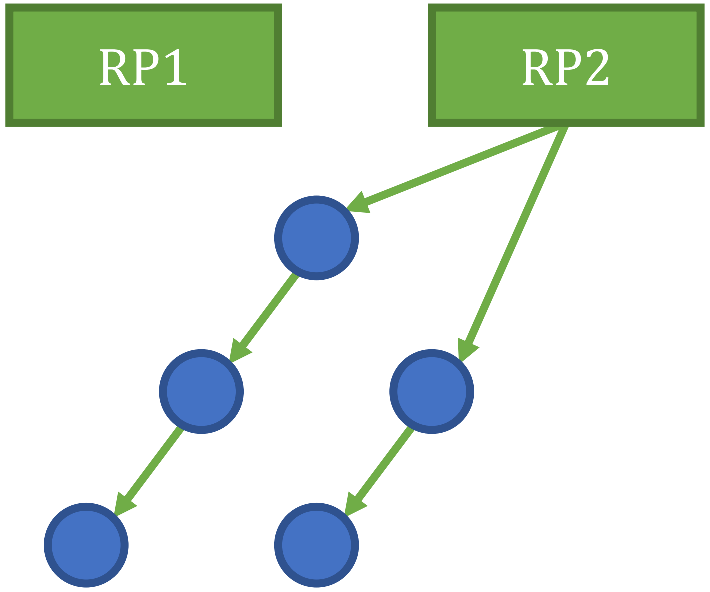
Core migrations

- Goal: seamless and reliable core migration
- Example: migrate a tree from RP1 to RP2
- RP1 notifies RP2 and subscribe to it; notifies the network to update NAME-RP mapping
- RP1 sends a marker packet M1 to all nodes in the tree
  - Nodes join the new multicast tree at RP2; keep the old paths toward RP1 to ensure reliable delivery
- With new tree established, RP1 sends a second marker packet M2, so nodes remove stale paths; RP1 also unsubscribe from RP2



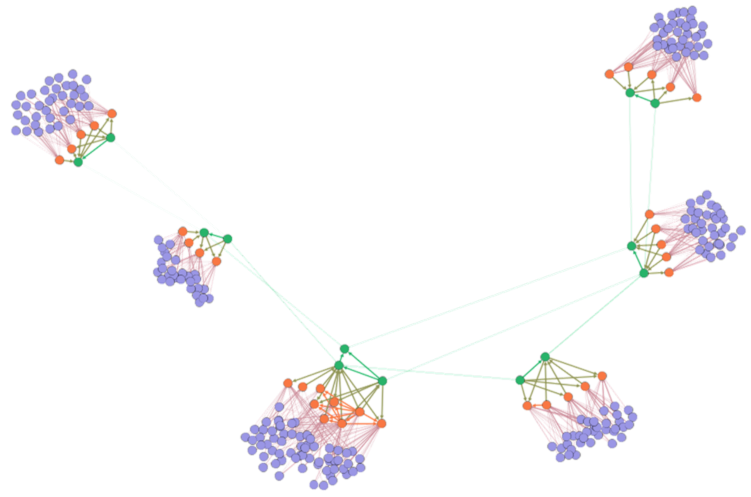
Core migrations

- Goal: seamless and reliable core migration
- Example: migrate a tree from RP1 to RP2
- RP1 notifies RP2 and subscribe to it; notifies the network to update NAME-RP mapping
- RP1 sends a marker packet M1 to all nodes in the tree
  - Nodes join the new multicast tree at RP2; keep the old paths toward RP1 to ensure reliable delivery
- With new tree established, RP1 sends a second marker packet M2, so nodes remove stale paths; RP1 also unsubscribe from RP2
- New multicast tree at RP2 established



Experimental results

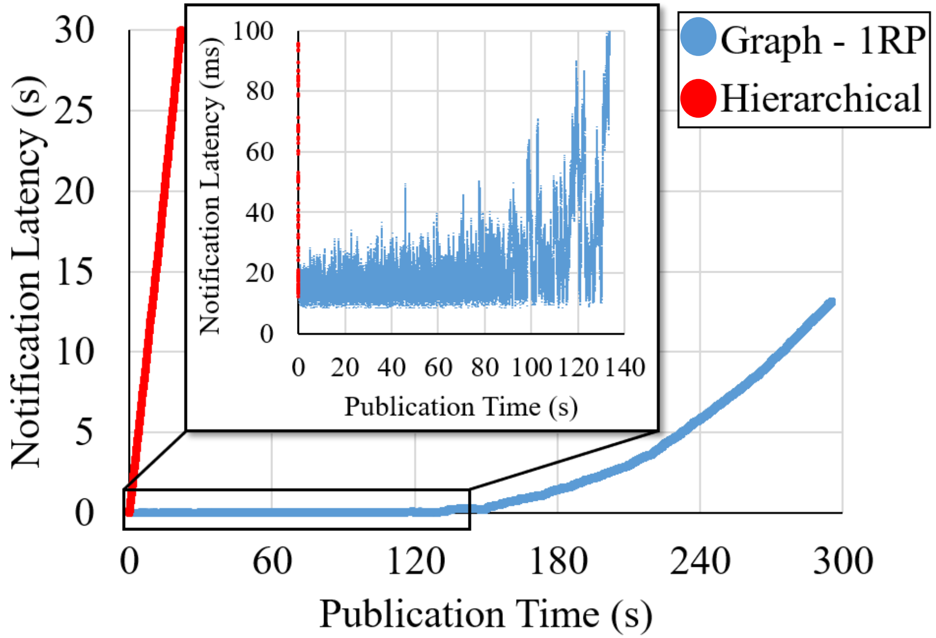
- Network simulation: evaluate the impact of POISE's design on latency, traffic and queuing
- Simulation setup
  - Topology with 277 routers
  - Namespace: disaster management from Wikipedia
    - 489 nodes, 732 edges (hierarchical equivalent: 1,468 nodes)
  - Subscribers: 6 per name, randomly placed
  - Publications: 514,620 pubs with Poisson distribution
    - Increasing rate: 1,500pkt/s – 2,000pkt/s
      - Increasing as disaster events unfold and more people involved
  - Notification latency and aggregate network traffic are key metrics



Network Topology (Rocketfuel 1221)

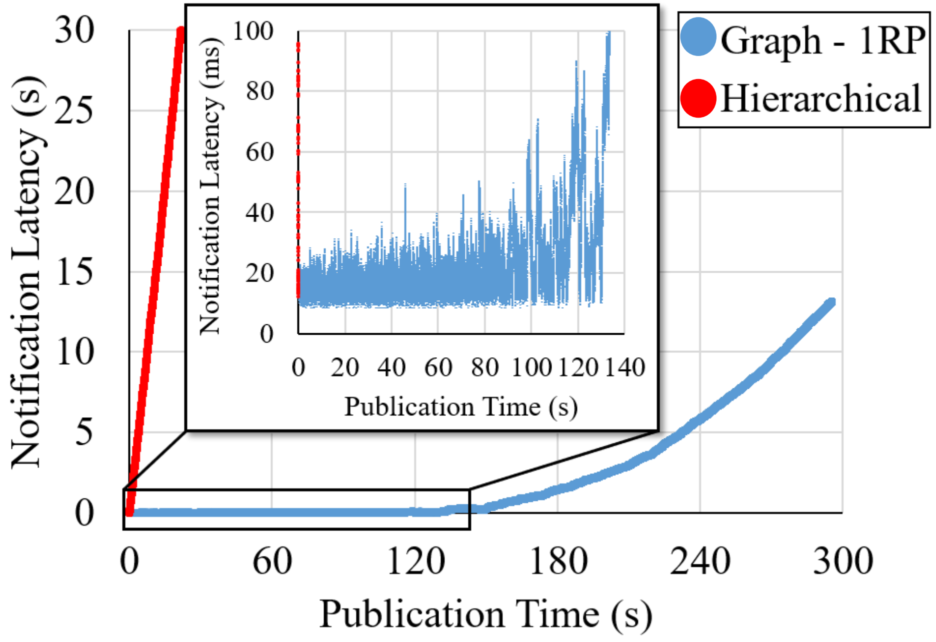
Experimental results

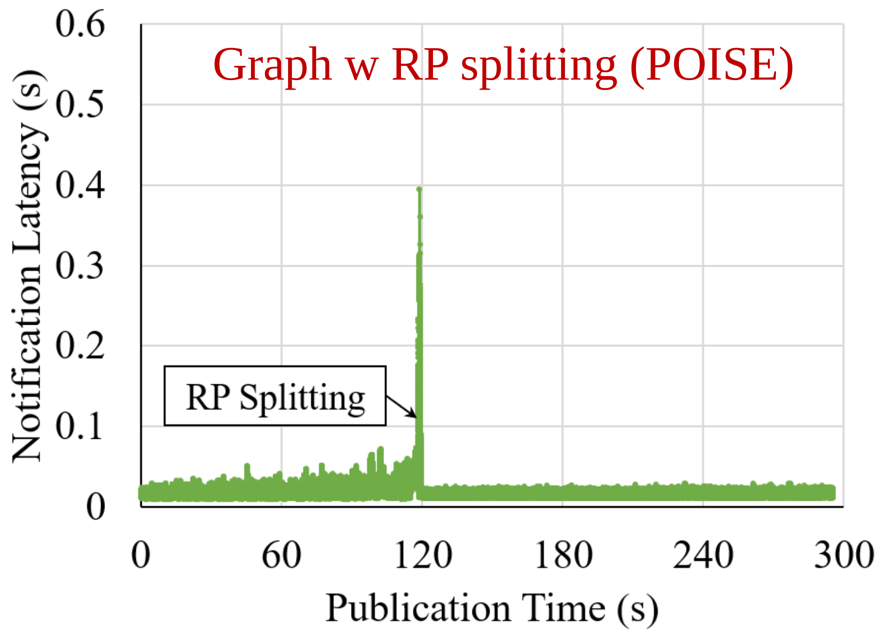
- Hierarchical namespace-based approach sees huge latency due to more publications caused queueing on the RP (red line)
- Graph namespace (even w/o RP partitioning) does a lot better (blue line)
- Graph namespace has low notification latency (<100ms) with low rate, but queueing is still observed when publication frequency gets higher



Experimental results

- Hierarchical namespace-based approach sees huge latency due to more publications caused queueing on the RP (red line)
- Graph namespace (even w/o RP partitioning) does a lot better (blue line)
- Graph namespace has low notification latency (<100ms) with low rate, but queueing is still observed when publication frequency gets higher
- Our solution (POISE) reduces the latency with sensible RP splitting

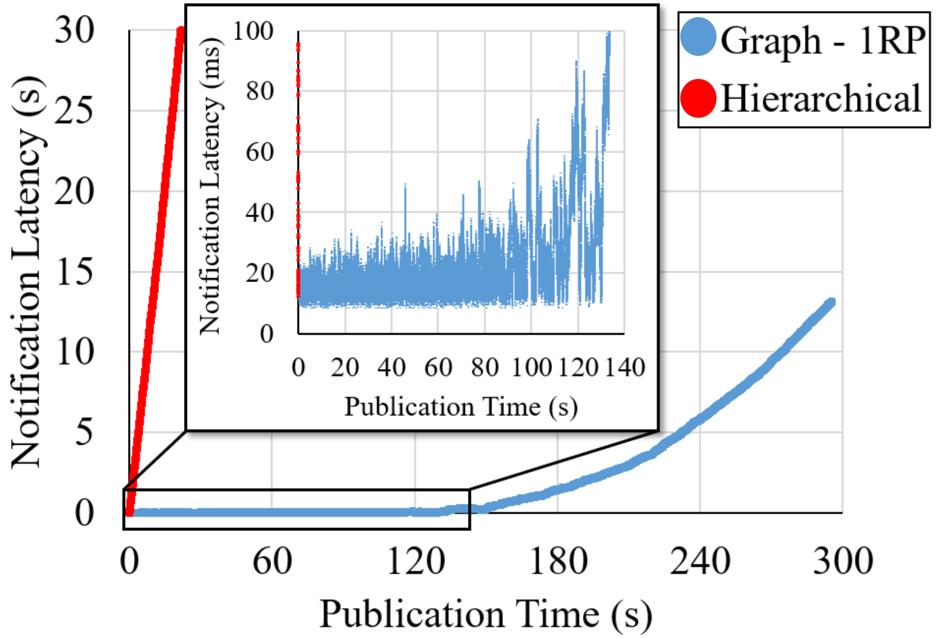


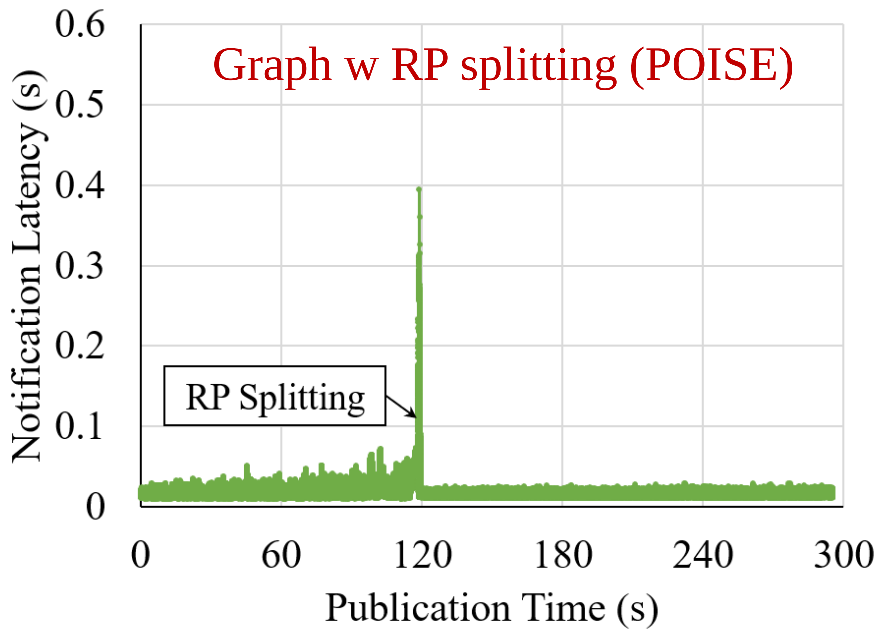


Experimental results

- Average latency of POISE is many orders of magnitude smaller
- Aggregate network traffic
    - Our solution (POISE) introduced a slightly higher traffic (<1%), to get the very low notification latency
    - Graph namespace reduces network traffic (by 41.41%) compared to hierarchical name-based approach

Solution	Avg. Notification Latency (s)	Aggregate Network Traffic (Gb)
Hierarchical name-based	247.742	866.27
Graph w 1 RP	2.741	483.08
Graph w RP splitting (POISE)	0.018	492.39

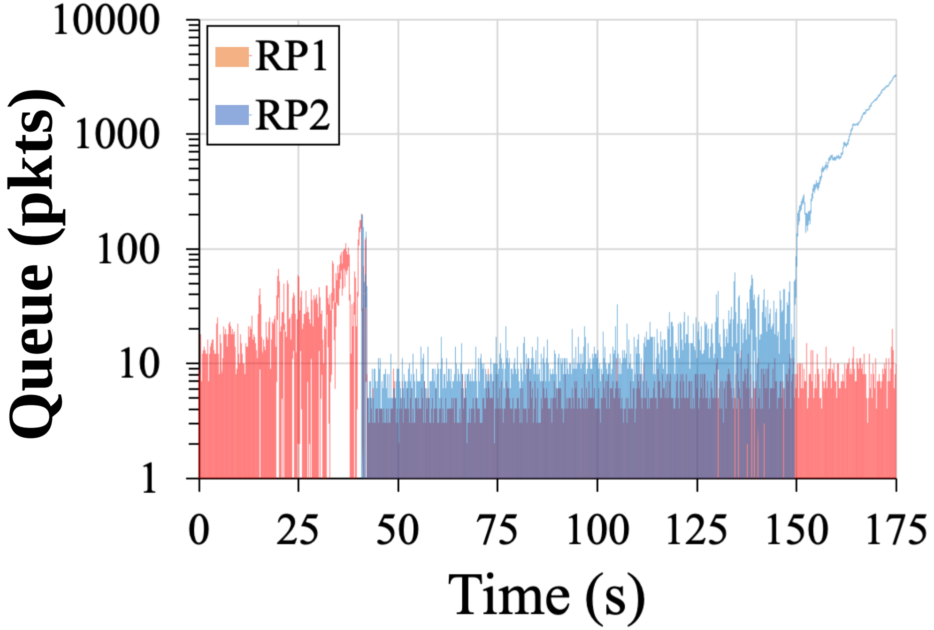




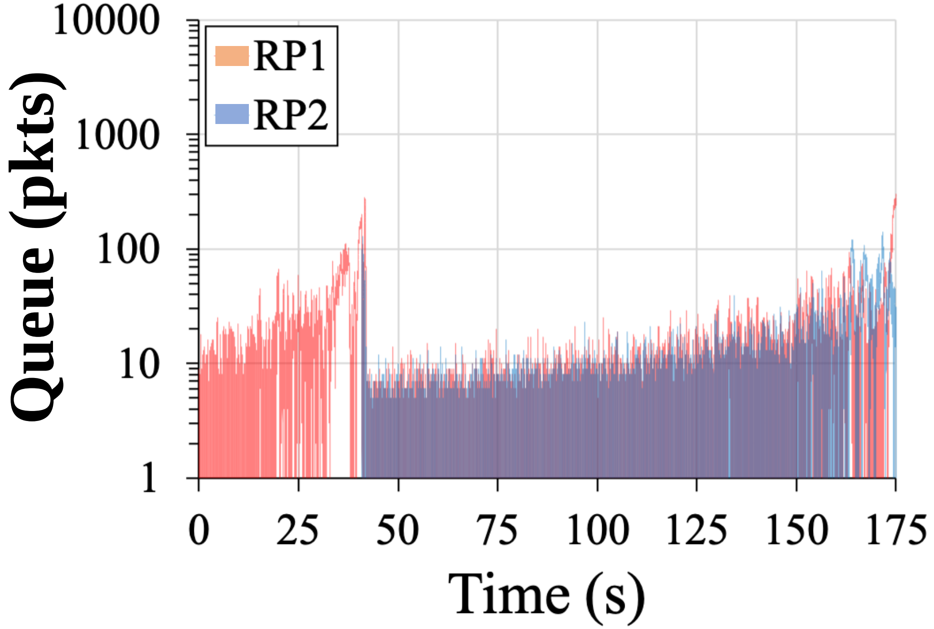
Experimental results

- Intensify the publication workload
  - To observe the difference in extreme traffic
  - 514,620 pubs with increasing rate:
  ~~1,500pkt/s – 2,000pkt/s~~ 1,500pkt/s – 3,500pkt/s
- Compare choice of graph partitioning
  - METIS
  - POISE: METIS+Tabu
    - Better queue size balance between two RPs

Load splitting w METIS

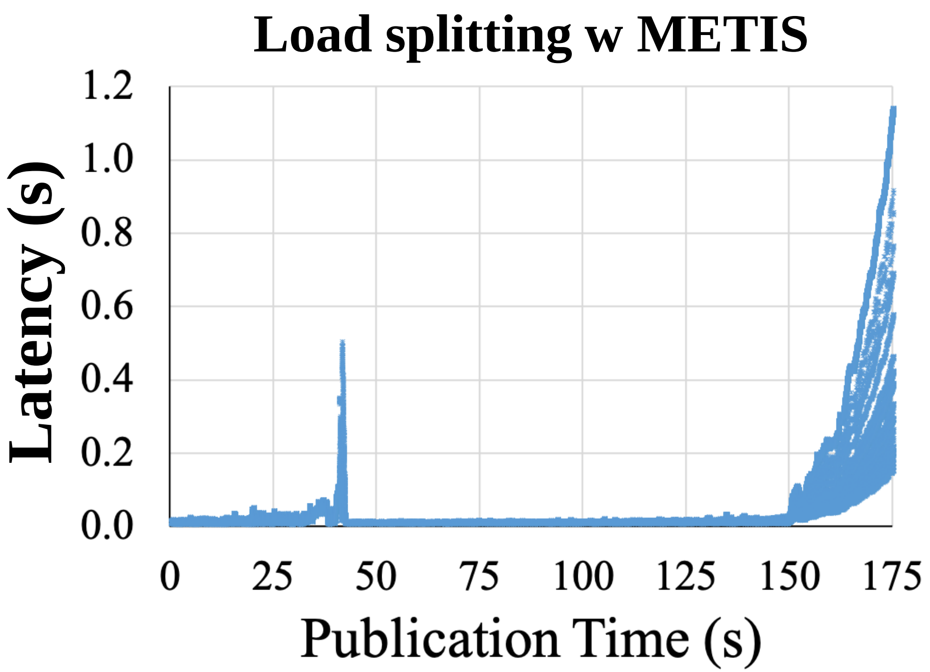


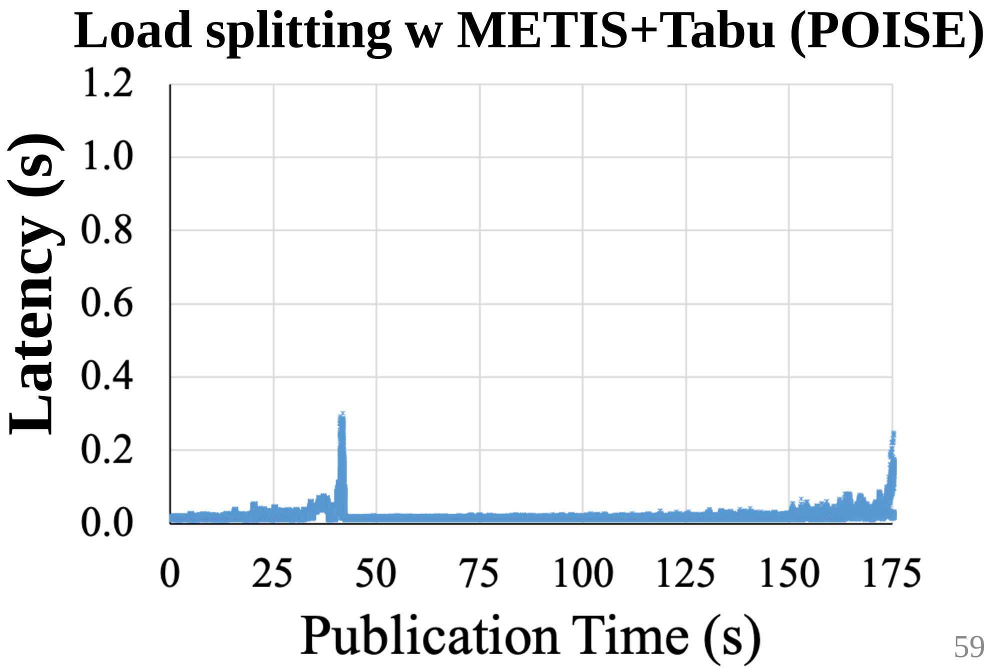
Load splitting w METIS+Tabu (POISE)



Experimental results

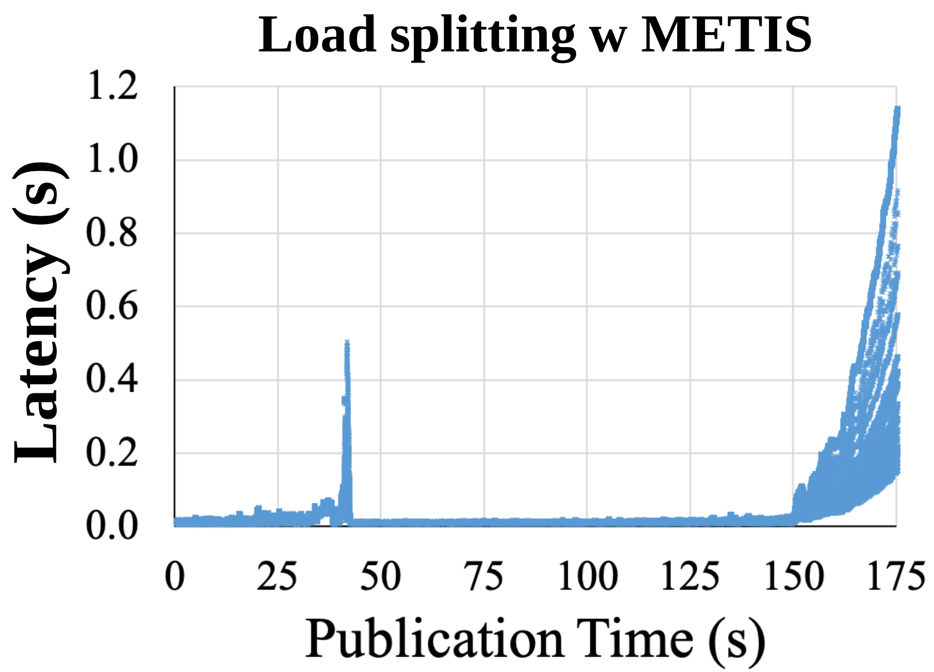
- Intensify the publication workload
  - To observe the difference in extreme traffic
  - 514,620 pubs with increasing rate: ~~1,500pkt/s – 2,000pkt/s~~ 1,500pkt/s – 3,500pkt/s
- Compare choice of graph partitioning
  - METIS
  - POISE: METIS+Tabu
    - Better queue size balance between two RPs
    - Better notification latency
      - Average: 0.396s vs. 0.583s

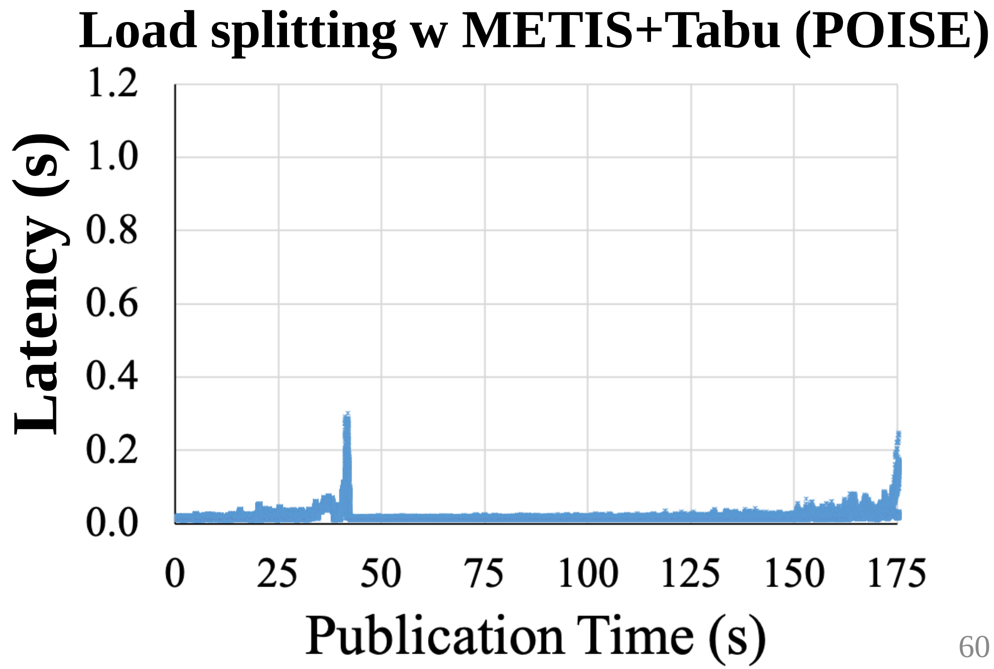




Experimental results

- Intensify the publication workload
  - To observe the difference in extreme traffic
  - 514,620 pubs with increasing rate:
  ~~1,500pkt/s – 2,000pkt/s~~ 1,500pkt/s – 3,500pkt/s
- Compare choice of graph partitioning
  - METIS
  - POISE: METIS+Tabu
    - Better queue size balance between two RPs
    - Better notification latency
      - Average: 0.396s vs. 0.583s
- Using this hybrid graph partitioning, POISE enables a load sharing with smaller latency and better balance





Summary

- POISE: Information dissemination enabling role-based pub/sub, supporting graph-based namespaces, with automatic load splitting --- use case: disaster management
  - POISE's Graph-based pub/sub outperforms hierarchical name-based pub/sub
  - POISE's graph partitioning outperforms METIS
  - POISE's RP migration is seamless and reliable