This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Graph-Based Namespaces and Load Sharing for Efficient Information Dissemination

Mohammad Jahanian, Jiachen Chen, and K. K. Ramakrishnan, *Fellow, IEEE*, *ACM*

Abstract—**Graph-based namespaces are being increasingly used to represent the organization of complex and ever-growing information eco-systems and individual user roles. Timely and accurate information dissemination requires an architecture with appropriate naming frameworks, adaptable to changing roles, focused on content rather than network addresses. Today's complex information organization structures make such dissemination very challenging. To address this, we propose POISE, a name-based publish/subscribe architecture for efficient topic-based and recipient-based content dissemination. POISE proposes an information layer, improving on state-of-the-art Information-Centric Networking solutions in two major ways: 1) support for complex graph-based namespaces, and 2) automatic name-based load-splitting. POISE supports in-network graph-based naming, leveraged in a dissemination protocol which exploits information layer rendezvous points (RPs) that perform name expansions. For improved robustness and scalability, POISE supports adaptive load-sharing via multiple RPs, each managing a dynamically chosen subset of the namespace graph. Excessive workload may cause one RP to turn into a "hot spot", impeding performance and reliability. To eliminate such traffic concentration, we propose an automated load-splitting mechanism, consisting of an enhanced, namespace graph partitioning complemented by a seamless, loss-less core migration procedure. Due to the nature of our graph partitioning and its complex objectives, off-the-shelf graph partitioning, e.g., METIS, is inadequate. We propose a hybrid, iterative bi-partitioning solution, consisting of an initial and a refinement phase. We also implemented POISE on a DPDK-based platform. Using the important application of emergency response, our experimental results show that POISE outperforms state-of-the-art solutions, demonstrating its effectiveness in timely delivery and load-sharing.**

Index Terms—**Information-centric networking, publish/subscribe systems, graph partitioning.**

I. Introduction

With large amounts of information generated, the relationships between content items are becoming more and more complex. Similarly, even organizations and environments involving people and entities, with their different roles and interests, have rich inter-relationship structures that go beyond simple hierarchical trees. This complexity can also be seen in networked environments, such as IoT, datacenter networks, containerized environments, and distributed file systems. Having a well-defined naming framework that is flexible enough to accommodate the needs of these complex structures is essential [1]. We have had to move from the simple and traditional hierarchical structures to more complex graph-based namespaces for organizing names. Example use cases we consider are disaster management, smart building with IoT, and resource, data management in clouds and distributed file systems. In all these cases, having a framework for efficient information dissemination, one that ensures all relevant actors receive the required information in a timely manner, is of importance and can be very helpful.

Publish/Subscribe (pub/sub) systems (*e.g.*, [2], [3]) are popular means of information dissemination today, and enable one/many-to-many push-based notification systems. Support for multicast in the network helps achieve efficient pub/sub by greatly reducing network traffic and server/client overhead, compared to server-based, poll-based pub/sub solutions [3]. Today, IP multicast is the prevalent multicast protocol, *e.g.*, in IPTV [4], albeit not in the multi-domain case. Despite its utility and efficiency, IP multicast has limitations which make it less than ideal for pub/sub in more complex and content-oriented pub/sub applications: it is tightly intertwined with IP multicast group addresses, does not capture the semantic relationships between groups (*e.g.*, the fact that one group publishes information that is in a subcategory of another group), and is limited by its IP address (sub-)space size. These shortcomings are problematic where there may be complex content/actor inter-relations with frequent churn in those relationships. This would put excessive burden on publishers and subscribers if IP multicast is used [3]. To overcome these challenges, we use a name-based multicast scheme for pub/sub [3], [5], leveraging the concept of Information-Centric Networking (ICN).

ICN [1], [6], [7] enables the network layer to understand content names, and provide forwarding and dissemination functionality independent of location, *i.e.*, IP addresses. This location-independent networking is very useful in a majority of networked applications, since often, it is the information that matters most to recipients rather than which point of attachment in the network it originated from [1]. Name-based pub/sub identifies a multicast group by its name, with the namespace capturing the relationship among those names [3]. There are two types of namespace design for name-based pub/sub: *topic-based* and *recipient-based*. In topic-based pub/sub (*e.g.*, [3]), a subscriber of a named topic, *e.g.*, "`/CaliforniaWildFires`", is interested in receiving all the content published at a finer granularity, *i.e.*, what is *under* that topic category, *e.g.*, "`/CaliforniaWildFires/WoolseyFire`'.

Manuscript received July 25, 2020; revised March 18, 2021; accepted June 2, 2021; approved by IEEE/ACM Transactions on Networking Editor V. Subramanian. This work was supported in part by the US Department of Commerce, National Institute of Standards and Technology under Award 70NANB17H188 and in part by the U.S. National Science Foundation under Grant CNS-1818971. *(Corresponding author: Mohammad Jahanian.)*

Mohammad Jahanian and K. K. Ramakrishnan are with the Department of Computer Science and Engineering, University of California, Riverside, CA 92521 USA (e-mail: mjaha001@ucr.edu; kk@cs.ucr.edu).

Jiachen Chen is with WINLAB, Rutgers University, North Brunswick, NJ 08902 USA (e-mail: jiachen@winlab.rutgers.edu).

Digital Object Identifier 10.1109/TNET.2021.3094839

1558-2566 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

In recipient-based pub/sub [5], a subscriber of a named role, *e.g.*, "`/fireDepartment/fireTeam1`", must receive all messages published to a coarser granularity, *i.e.*, sent to every other role *above* that role when it comes to authority or responsibility, *e.g.*, to "`/fireDepartment`". Unifying these two types, we support both topic-based and recipient-based pub/sub.

Namespace design is an integral part of ICN. State-of-the-art ICN architectures, namely Named Data Networking (NDN) [1], [6] supports strictly hierarchical namespaces, implemented as prefix trees [8]. This hierarchical structure falls short in efficiently modeling complex, multi-dimensional namespaces [9], such as today's increasingly complex information organization structures, where a role (node) can have many dimensions (parents), *e.g.*, function, location, time, *etc.* The dynamic nature of these structures in many scenarios, makes these namespaces even more challenging to manage. While it is possible to convert a graph to a strict hierarchy, it will lead to huge amount of redundancy, thus making the management and modification of the namespace extremely costly and inefficient. A study on a Wikipedia dump in [9] shows that converting a typical graph-structured namespace with 10^6 categories (graph nodes) to its hierarchical equivalent, would result in 6.07×10^{54} categories.

We propose POISE, designing dynamic graph-based namespaces for in-network name-based pub/sub, with the introduction of an information layer to manage it. Although POISE supports both topic-based and recipient-based pub/sub, our particular focus here is on recipient-based pub/sub. Additionally, we propose a rendezvous point (RP)-based pub/sub protocol, with the dissemination logic following the graph-based namespaces, to deliver all relevant information to their intended/required recipients (mainly first responders) in a timely manner using push-based multicast. POISE handles possible cycles in the graph through preventive DFS-based cycle detection in the graph, as well as data plane nonce-based loop detection [8]. We share the workload among multiple RPs, where each RP is responsible for managing a subset of the namespace graph and functions as the core of the subscription trees associated with those names. Often in many real-world scenarios, the workload-per-RP distribution is non-uniform and difficult to predict. In an RP-based pub/sub, this could cause an excessive load on one RP managing names corresponding to those more intense workloads, thus making it a "hot spot"; an example of this may be in a multi-player online gaming environment, as in [10]. While it is practically infeasible to optimally balance the load across the whole network (due to the amount of frequent periodic communication needed which is especially difficult with large and/or bandwidth-limited networks), we eliminate the traffic concentration by automatically splitting a congested RP's (*i.e.*, hot spot's) workload: the RP partitions its namespace (sub-)graph with the objective of finding two balanced segments while minimizing inter-RP communication, decides which names to relinquish, and triggers the migration of subscription tree cores related to those names to a new RP or an existing under-loaded RP. Our graph partitioning problem involves calculation of weights for vertices and edges. However, due to the nature of our partitioning formulation, these weights depend on the cut itself; thus making our objective function a "complex" one [11]. As a result, off-the-shelf graph partitioners such as the popular tool METIS [12] (as well as its parallelized version, ParMETIS [13]) fall short. To overcome this, we propose a hybrid splitting procedure consisting of a heuristic (METIS) and meta-heuristic guided search refinements (using Tabu Search [14]). Our results show the effectiveness of our design. While we consider a number of example use cases of POISE in different environments, we primarily focus on the application of POISE to information dissemination among first responders in disaster scenarios, an application that requires timely information delivery.

The key contributions of this paper are the following: 1) A recipient-based pub/sub framework with automatic load splitting for efficient information dissemination. 2) Support for free-form (*i.e.*, not limited to a particular structure such as hierarchy) graph-based namespaces and an information layer to capture rich information organization structures; our simulation results show that this is more efficient than using state-of-the-art hierarchical namespaces. 3) An automatic name-based and workload-driven, novel hybrid graph partitioning procedure and load splitting along with a seamless and lossless core migration mechanism; our results show the effectiveness and correctness of our core migration, and improved quality and resulting network efficiency of our partitioning procedure, compared to popular off-the-shelf graph partitioning tools. We further demonstrate the benefit of using our Tabu search-based refinement compared to an iterative implementation of METIS, as well as ParMETIS with Adaptive Repartitioning. 4) An implementation of a POISE RP including its graph-related operations on a DPDK-based platform; our micro-benchmarking shows that the overhead of graph-based operations justifies using our RP-based solution rather than name-expansion at every hop.

II. Background and Related Work

Information-Centric Networking (ICN) [1] enables access to named objects, independent of their locations. In ICN, contents and entities can be named through identifiers. Having a network layer that recognizes these identifiers can help deliver information without separately establishing an end-to-end communication channel, support in-network content caching, aggregate queries, and provide content-oriented security. All these result in more efficiency, compared to traditional host-centric networks, such as IP. Two notable ICN architectures are NDN [6] and MobilityFirst (MF) [7]. While our proposed information layer can work on top of any ICN or IP architecture, we focus on ICN as it is more efficient for our name-based pub/sub [15].

Publish/Subscribe (pub/sub) has become a widely used, popular service over the Internet, in form of RSS feeds, online social networks, *etc.* Most popular pub/sub solutions today are server-based; where subscribers either poll a logically centralized server (via HTTP), or a long-lived connection for timely delivery is maintained [16]. These approaches can be limited in scalability. Broker-based solutions (*e.g.*, ONYX [17]) use an overlay network with distributed brokers, and avoid traffic

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

concentration. However, the dependency of these solutions on XML data and assertions to decide forwarding paths, couples the information structure with the network layer, making the forwarding function complex. Having pub/sub in the network can help with scalability, and has been proposed for ICNs [2], [3], [18]. ICN with push-based publish/subscribe service models [3] have been proposed. COPSS [3] enhances the query/response model of NDN by allowing consumers to issue a long-standing request, *i.e.*, subscription, for all content related to (subsets of) a name, whenever they are published. It can outperform IP multicast, poll-based methods and flooding-based broadcast [3], [19], in terms of aggregate network load and latency. CNS [5] extends COPSS by introducing recipient-based pub/sub, that can help with information dissemination. Our work moves a step further by relieving the strict hierarchy restriction to enable complex free-form graph-based namespaces.

Graph-based information organization has been gaining attention and shown to be important because of its richness and efficiency compared to the more traditional hierarchical structures across multiple application domains. Wikipedia is a very popular and notable example of an information organization system designed as a graph structure: each article can belong to a number of categories, *i.e.*, dimensions [20]. Graph structures for information have been proposed and used in many other contexts as well, *e.g.*, databases [21]. cloud computing [22], and file systems [23]. These works have primarily focused on information organization for storage and indexing. Our work focuses on in-network information organization for name-based information dissemination, extending the current hierarchical structure of NDN [6] to a graph-based one.

Graph partitioning is an important graph operation, and has been an area of research for decades. Optimal graph partitioning is considered to be NP-hard [24], so solutions based on heuristics and approximations exist. A very well-known partitioning method is multi-level partitioning [25] (and its tool METIS [12]), which using heuristics, coarsens the graph, does an initial partitioning, and then uncoarsens it. METIS has been widely used for load splitting and balancing in various contexts [26], [27]. The parallel version of METIS, ParMETIS [13], achieves speedups using message passing interface (MPI)-based parallel processing. ParMETIS also includes additional routines for adaptive re-partitioning, to enable fast incremental updates to an already-existing partitioning, in case of weight changes in the graph, rather than complete re-partitioning from scratch. Some methods use the streaming graph partitioning approach which is used to process partitioning a piece of data on the fly, *e.g.*, [28]. These algorithms are very fast but their solution quality is lower. This approach is most suitable for extremely large graphs (in the order of trillion vertices). There are approaches using iterative improvement methods for graph partitioning. These methods typically provide high quality solutions. A bad choice of the iterative method and its parameters can make the procedure slow. Work in [29] proposes a graph partitioning algorithm using Tabu search, and shows that it outperforms another popular meta-heuristic method, Simulated Annealing [30], regarding both solution quality and timeliness. Sometimes the objective of partitioning is more than a simple sum of weights, and can be a complex function of the cut itself. This is characterized as the "chicken and egg problem" in [11], as the objective function needed for partitioning decision must be calculated after the partitioning is done; ours belongs to this class. Approaches to solve this class of problems have been proposed in works such as [31]–[33] for specific cases. The work in [11] justifies the use of standard partitioning as a good starting point, and then perturb it during the iterative refinement procedures.

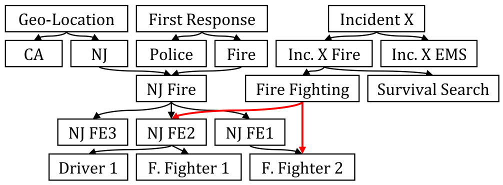


Fig. 1. Graph-based namespace: incident command chain example.

Multicast core migration aims at moving a core-based multicast tree [34] from one core (RP) to another, for better load balancing or failure resiliency. Traditional core migration works reported in [35] focus on IP multicast, and primarily focus on low-level migration criteria such as link or node utilization, *etc.* We focus on criteria pertaining to name-level workload in our work. Work in [10] shows the need for RP migration in name-based multicast, but only uses a random load splitting mechanism. We improve it by formulating the workload splitting scheme through a rigorous workload-driven graph partitioning algorithm along with a migration procedure that is reliable and loss-less.

III. Use Cases of POISE

POISE is applicable in a variety of different contexts, where a pub/sub communication model is needed. Its biggest benefit is in cases where the dissemination is done according to a complex, multi-dimensional namespace, and timely delivery of relevant information to all intended recipients is required [36]. In this section, we illustrate how POISE can specifically help with managing sample namespaces for several use cases.

A. Disaster Management

During disasters, a large amount of information is disseminated, and it involves distributing it to many participants such as incident commanders, first responders, volunteers and civilians. ICN and name-based pub/sub have been shown to be very suitable and beneficial for disaster scenarios [5], [37]. Furthermore, due to the complex nature of today's command chains of first responders, a graph-based naming framework (an example disaster management namespace is shown in Fig. 1, with more detail in §IV-B), is needed to represent the multiple reporting hierarchies. Furthermore, in an RP-based framework for name-based pub/sub for disaster management, overload and hot spots (as mentioned in §I) are highly likely to occur. In a given disaster, particular roles (*i.e.*, names) may receive much higher demands than other roles (*e.g.*, 'firefighting' roles in a wildfire incident). This motivates our RP splitting and graph partitioning to eliminate traffic

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

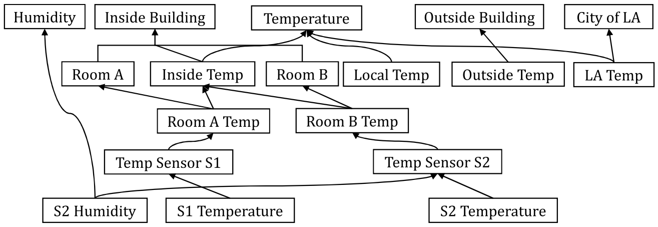


Fig. 2. Example namespace for IoT HVAC system.

concentration. Furthermore, a major problem often caused by disasters (especially natural disasters), is that the network and servers can experience excessive load and congestion, making many services unavailable [5]. Our work addresses this by creating an efficient information organization and load sharing framework that dramatically reduces network load during disasters. Works such as [38], [39] have been proposed to leverage delay/disruption-tolerant routing with ICN in disasters in case of complete or partial infrastructure failures. These works, while orthogonal to ours, can be leveraged by POISE. POISE's information layer can run on top of such delay-tolerant protocols in disconnected environments.

B. HVAC System: A Smart Building With IoT

The use of Internet of Things (IoT) for building smart cities, buildings and homes is becoming increasingly more popular. The interactions between different elements in an IoT environment, *i.e.*, sensors, actuators, *etc.*, can be complex. Fig. 2 shows a small (partial) namespace for a HVAC (Heating, Ventilation, and Air Conditioning) system in a building as an example use case for POISE. Information can have multiple dimensions: type of information (*e.g.*, temperature or humidity), location of the information (*e.g.*, inside the building or room A), *etc.* The naming schema used in current name-based architectures for IoT (such as NDN RIOT [40]) use strictly hierarchical structures, which falls short in efficiently capturing such complex multi-dimensional structures. Using POISE's graph-based namespace (such as in Fig. 2), we can support a publish/subscribe (topic-based) capability to support many-to-many delivery from sources (sensors) to destinations (actuators). Sensors publish to names; a new publication can be generated periodically, or if there is a substantial update to report. All subscribers of that name and all the names reachable from it will receive it. Different actuators (*e.g.*, heating or cooling) can choose different granularity levels based on their settings, subscribing to "`Temperature`" (to get every temperature reading from any source: its own sensor, inside building, outside building, and externally from city of LA news report), subscribing to "`Room B`" (to get any information in Room B: temperature, humidity, *etc.*). By subscribing to "`Inside Temperature`", an actuator will receive all temperature readings recorded in Room A or B, whenever they are published.

C. Resource Management in Clouds

The use of virtualized container environments for cloud services have become popular in the past decade. In large-scale systems, they can involve the interaction across many users and resources, warranting a scalable data structure and communication system of managing their orchestration. Kubernetes [22], as a prominent example, defines namespaces for resource orchestration. With POISE, we can have a generalized graph-based namespace (with nesting) that captures all the different resources and components such as clusters, projects, and libraries, while allowing a systematic push-based pub/sub notification of changes to relevant users. Fig. 3 shows a (partial) POISE namespace as an example for such environment. The edge directions denote the flow of information. For example, upon any change in "`Database`", an immediate notification will be sent to all subscribers of "`Development A`" (as well as other ancestors of "`Database`"), which are members of a team working on the development of "`Project A`", and using the "`Database`" cluster. This is especially helpful at an enterprise-level with a large complex namespace. The authorization and access can be captured with the directed edges. Any publication made to a name can be generated by any user that has the authorization to make changes (*e.g.*, modification of data or code) to the resource(s) associated with that name. The POISE namespace captures a flexible integration of isolation and sharing; *e.g.*, in Fig. 3, "`Pod 1`" is hidden from anyone not associated with "`Project A`", while "`Pod 3`" is visible to members of both "`Project A`" and "`Project B`".

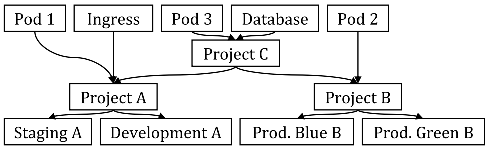


Fig. 3. Example namespace for containerized platforms.

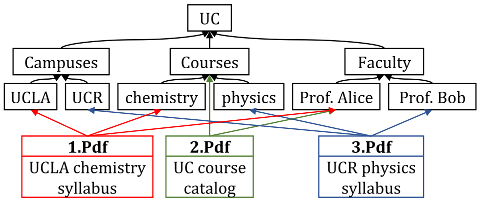
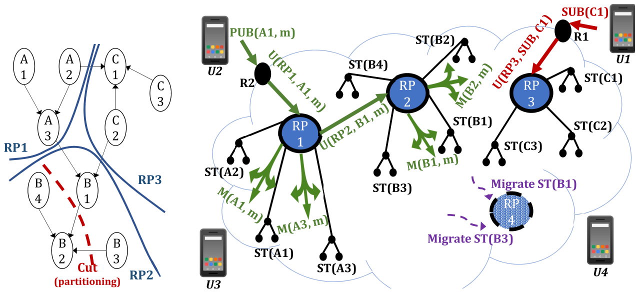


Fig. 4. Example namespace for distributed replica management.

D. Data Replica Management in Distributed File Systems

Distributed file systems are widely used in cloud systems, where each server in the datacenter hosts files belonging to parts of the overall directory. Work in [41] shows the benefit of using ICN for distributed file systems in a datacenter. A graph-based namespace for the directory system can be more efficient than just a hierarchical tree structure. Fig. 4 shows a small example of such a namespace for a university system's distributed file system. A strictly hierarchical version of the namespace graph would lead to a large number of duplicate nodes; for example the file "`1.pdf`" would have at least three name hierarchies, one for "`/UC/Campuses/UCLA/1.pdf/`", one for "`/UC/Courses/chemistry/1.pdf`", and one for "`/UC/Faculty/Prof.Alice/1.pdf`'. For scalability and reliability, distributed file systems typically require replicas of files in different data servers. For example, the Hadoop

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Namespace (b) Information dissemination architecture

Fig. 5. A schematic overview of the architecture of POISE.

Distributed File System (HDFS) [42] requires exactly 3 replicas of a given file chunk (in default mode). As files can be modified, consistency of data among replicas becomes an issue. POISE can make the management of data replicas quite convenient and efficient. For example, server D1 hosting all the files under the courses category, will subscribe to "`Courses`". Server D2 hosting all the physics files, will subscribe to "`physics`" (which is under "`Courses`"). Any modifications to a file belonging to physics, *e.g.*, file "`3.pdf`" will be published to the name "`physics`", thus both D1 and D2 will receive the notification, because the published update will be propagated along the name paths in the namespace, and will apply the changes to their stored replica files (or caches). This will provide flexibility for replica management, enabling multiple ways for different replicas' sub-directories to overlap.

IV. Architecture and Design

This section presents the overall architecture of POISE, its naming and pub/sub design details. In the remainder of this paper, we use the disaster management use case as an example to explain the various aspects of POISE.

A. Overview of POISE

An overview of POISE's architecture is shown schematically in Fig. 5. POISE's namespace supports free-form graph structures, as shown in Fig. 5(a), rather than being restricted to the state-of-the-art hierarchical namespaces [43], [44]. This enhancement is possible through decoupling of *information layer* (which manages names and their relations in their natural form, supporting complex graphs) and the *service layer* (which manages the names used for name-based forwarding at every ICN router), which are coupled together in current Named Data Networks [6]. Each vertex in the graph in Fig. 5(a) is a name, *i.e.*, a role or attribute, and the edges show relations among them. POISE's information dissemination framework is a name-based pub/sub [5] with the support of name-oriented core-based multicast, with *rendezvous points* (RPs) being the cores for groups; each name also identifies a multicast group. In addition to being the core for the multicast tree (similar to traditional PIM-SM [34], NDN COPSS [3], or MF multicast [18]), POISE's RPs operate at the information layer; in other words, they are information-layer-enhanced RPs. As shown in Fig. 5(a) and Fig. 5(b), the namespace is shared among three RPs (*i.e.*, RP1, RP2, and RP3), each RP managing the sub-graph it is responsible for, and maintaining the subscription trees associated with each of the names (groups) it is hosting; *e.g.*, RP1 is the core for the subscription tree (ST) for $A1$, $A2$, and $A3$. A *name-to-RP* mapping resolution service resolves a name in the namespace to the RP it is hosting; *e.g.*, the name $C1$ would be mapped to RP3.

The subscription path is shown in Fig. 5(b) (in red); user $U1$ wants to subscribe to $C1$ (which implicitly means subscribing to all ancestors of $C1$ in the namespace as well). $U1$ sends this request as $SUB(C1)$, without the need to know which is the associated RP or where it resides. $U1$'s first hop router R1 performs the resolution and relays the request as a unicast (U) message to the correct RP, *i.e.*, RP3; thus, $U1$ *joins* $ST(C1)$. The publication path is also shown (in green); $U2$ wants to publish message m to all subscribers of name $A1$ (which implicitly means publishing to subscribers of all descendants of $A1$ as well). $U2$'s first hop router relays this publication as a unicast message to its corresponding RP, namely RP1. Expanding $A1$ to its descendants (*i.e.*, name expansion), it sends m as a multicast (M) downstream to $ST(A1)$ as well as $ST(A2)$. Additionally, RP1 recognizes that there is an edge leading from $A1$ towards a name outside RP1. Thus, RP1 sends a unicast message for this name, $B1$ to its RP, RP2. Note that RP1 only has visibility of namespace up until $B1$ and not further. Performing a similar name expansion, RP2 processes the received request by going through its namespace, which leads to multicasting m downstream along $ST(B1)$ and $ST(B2)$. Thus, users for both subscription and publication scenarios need only send *one packet*, destined to only *one name*; the network takes care of expanding the packet to additional names, if needed. More details on the information layer design and pub/sub dissemination are provided in [36].

Each RP's workload has a correlation with the part of the namespace it is managing. However, additionally, the load-per-name distribution is likely to be non-uniform, and hard to predict. To address this, another important feature of POISE, *automatic load splitting* is performed to eliminate traffic concentration. Consider the case when RP2 encounters a large amount of workload exceeding its threshold, thus making it a *hot spot*. Triggered by this, RP2 will perform a *partitioning* procedure on its own sub-graph, to provide two balanced *segments*, shown as *Cut* in Fig. 5(a). It thus decides to keep $B2$ and $B4$, and relinquish $B1$ and $B3$ to another RP (*e.g.*, RP4), which can be a regular ICN router configured to be a new RP for this environment. As a result, the subscription trees for $B1$ and $B3$ will be migrated to RP4, via a *core migration* procedure. The name-to-RP mapping will then be updated accordingly (§V).

B. Information Layer and Graph Namespace

POISE supports free-form graph namespaces with their natural structure for in-network information-centric dissemination, without the need to restrict them to any particular data structure, such as a hierarchy or prefix tree as NDN [6], or NDN-based solutions such as CNS [5] do. Using the disaster management use case as example, let us consider Fig. 1, a simple namespace of an incident management command structure. As we can see, it does not follow a strict hierarchy, but a graph.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

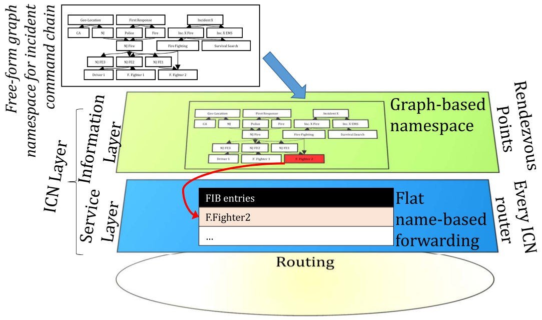


Fig. 6. ICN layer design in POISE.

Support for graph namespaces in information dissemination is made possible in POISE through a decoupling in the ICN layer and the introduction of information layer, as shown in Fig. 6. Only RPs (designated ICN routers that perform name expansion) need to understand and maintain name relationships in the graph. This design choice brings a number of significant benefits: it makes the ICN-layer namespace simpler and smaller, leads to smaller FIB tables, and eliminates redundant messages used for subscription and publication. More details on the information layer and the comparison are elaborated in [36].

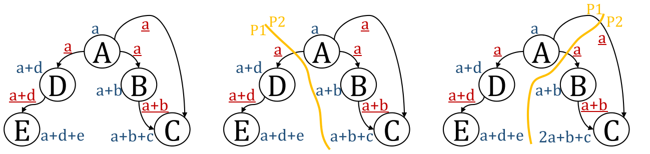
C. Recipient-Based Pub/Sub

POISE enables recipient-based pub/sub [5], enhanced with an information layer supporting graph-based namespaces. In this name-based pub/sub, subscribing to a name means implicitly also subscribing to a set of names related to that specific name, in accordance with the namespace. POISE's pub/sub logic follows the command chain graph-based namespace. In the case of disaster management, first responders and volunteers subscribe to (listen to) names, and civilians and incident commanders publish to names. Given the namespace in Fig. 1, subscribing to "`F.Fighter2`", implicitly means also subscribing to all of its ancestors, *i.e.*, "`NJ FE1`", "`FireFighting`", "`NJ Fire`", *etc.* Conversely, publishing to "`NJ Fire`", implicitly means also publishing to all of its descendants, *i.e.*, "`NJ FE1`", "`NJ FE2`", "`F.Fighter2`", *etc.* Expanding a name to all of its descendants on the publication path according to the namespace graph, is performed by the RPs, in a load-shared way. This design is beneficial where dynamically-formed interacting groups and individuals involved need to be notified with messages relevant to their tasks in a timely manner, whenever they are published or available, making sure maximum coverage and accuracy is achieved. Details of this protocol exchange are in [36].

V. Automatic Load Splitting

A. Partitioning Namespace Graphs

POISE's namespace graph partitioning aims at distributing the load among RPs if traffic concentration overloads an RP. Partitioning is performed locally on the congested RP, only on the (sub-)namespace that it is hosting, dynamically. We mainly use the monitoring of the recent queue size at RPs to measure its load, and use the recent multicast and unicast workloads (explained below) to label the graph for partitioning.



(a) Before partitioning (b) Partitioning 1 (c) Partitioning 2

Fig. 7. Partitioning impacts multicast workload weight of names.

1) Problem Description and Solution Overview: We leverage graph partitioning algorithms to determine which part of the namespace should reside at which RP for load splitting. We treat the namespace as a directed graph with weights (labels) on vertices (*i.e.*, names) and edges. The initial (input) vertex weights represent messages sent to each name explicitly from publishers (we call it *incoming unicast load*). To determine the number of messages multicast from a node (called *multicast workload*), we need to consider the incoming unicast load from all of its ancestors. The weight of the edges going out of a name are set to be the multicast workload of that name. The total weight of the edges going out of an RP to other RPs represents the total amount of outgoing inter-RP communication (which we call *outgoing unicast load*). We try to balance the sum of multicast workload and outgoing unicast load, in the two partitioned segments and seek to minimize the cut cost. A complexity here is that the decision of the partitioning can alter the weights, *i.e.*, "the chicken and egg problem" [11], thus making the off-the-shelf graph partitioners inadequate; we explain this with an example.

Fig. 7 shows a simple namespace graph at different stages. Let us denote the incoming unicast load of each name (node) as a, b, c, *etc.* Assuming no partitioning (*i.e.*, whole namespace in same RP), the multicast workload of each name is shown in Fig. 7(a) (blue labels next to each name). *E.g.*, name `C` has to send out publications related to itself, `A`, and `B` to its subscribers; thus making its multicast workload $a+b+c$. Edge weights in Fig. 7, denoting the RP-to-RP communication on that link, in case it gets cut, is shown in red and is underlined.

Considering the graph shown in Fig. 7(a), there may be multiple ways to partition a graph. Two examples are depicted in Fig. 7(b) and Fig. 7(c). As seen in the figures, the result of multicast workload of name `C` (and thus the total cost of the resulting graph) differs in the two figures; it is $a + b + c$ in Fig. 7(b) and $2a + b + c$ in Fig. 7(c). The one extra message `C` receives from `A` in Fig. 7(c) is due to the fact that in this scenario, `B` relays what it has received from `A` to `C`, not knowing that `A` is also a parent of `C`; while in Fig. 7(b) the graph cut is in a way that it does not cause such duplication. This shows that the graph's multicast workload weight values are a function of partitioning itself; thus, a standard partitioning tool with fixed weights is not sufficient to solve our partitioning problem. While such static methods can provide a fast, scalable partitioning solution, they do not achieve a sufficiently high-quality and come close to optimality, as they do not take into account the weight changes due to the cut. To address this, we propose the use of a hybrid approach of heuristics (classic static graph partitioning) followed by a refinement period. This refinement is an iterative procedure that dynamically adapts to weight changes as it progresses. One possible approach for it is to devise an iterative approach with successive runs of METIS, where at each iteration, the new weights based on the previous cut is fed back to METIS. Thus, at each iteration the best partitioning so far is returned as the solution of that iteration. A more efficient variation of this approach is to use ParMETIS's adaptive re-partitioning routine [13] at each iteration, rather than running METIS from scratch each time. While both these approaches may improve upon the solution quality of METIS, they are prone to getting stuck at local optima at an early stage, and also in not exploring the best possible search paths. To address this issue, we use Tabu search for the refinement period in POISE, as it provides a more thorough, but guided search as a meta-heuristic, to iteratively improve on the initial result provided by METIS. While out algorithm supports k-way partitioning, we focus on bi-partitioning in this paper (refer to [36] for details.)

As described earlier, we pay special attention to the quality of the partitioning solution, as a high-quality partition significantly reduces the resulting network traffic overhead and latency in POISE (demonstrated with results in §VI). Given that: 1) our partitioning is only performed occasionally and only upon RP overload (instead of all the time), 2) is run locally (rather than coordinating across multiple RPs), 3) concerns itself with balance among the two graph segments within an RP (rather than across the whole network), and, 4) deals with graph sizes of moderate sizes (in the order of hundreds or thousands of vertices, rather than millions, for each separate graph connected component, that is input to a partitioning pipeline); we believe it is reasonable to favor partitioning quality more over scalability. That said, we refrain from using brute-force approaches and instead use the parameters of our proposed algorithm in a way that produces high-quality results with the least amount of processing overhead. Also, in cases where the graph namespace consists of a number of separate connected components, each connected component sub-graph can be processed for partitioning independently and concurrently, thus enabling faster and more scalable computation in POISE.

2) Theoretical Foundations: In this sub-section, we formally define and mathematically represent the weighted namespace graph for partitioning, and how weights are calculated.

Multicast Workload (MW). An important part of our weight calculation, is calculating $MW(v_j)$ for each vertex v_j. Its value contains the aggregation of the vertex's ancestors' incoming unicast loads, reaching vertex v_j over all possible paths. When traversing within an RP for propagation, *e.g.*, using DFS traversal, only one path needs to be counted for multicast. On the contrary, if traversing across graph cuts (inter-RP), every path with a unique input-output pair of vertices needs to be counted. The value of $MW(v_j)$ depends upon the amount of load that v_j experiences, caused by every other vertex. Thus, we need to solve "how many times the incoming unicast load from v_i is received at v_j", which we call the *duplication factor*. We describe these in further detail.

Let us define a *simple path* as a path between two vertices that *does not* include a cycle fully contained in one segment. We then define P_{ij} as all simple paths from v_i to v_j:

$$P_{ij} = \{p_{ij}^1, p_{ij}^2, \dots\} \tag{1}$$

where p_{ij}^k is the k-th simple path from v_i to v_j, and is an ordered tuple of vertices, formed as $p_{ij}^k = (v_i, \dots, v_j)$.

In a partitioned, *i.e.*, cut graph, we define a *Cut* C as the multi-set of edges cut by partitioning:

$$C = \{(v_i, v_j) \mid part(v_i) \neq part(v_j)\} \tag{2}$$

where $part(v_i)$ denotes which graph part (segment) the vertex v_i belongs to, after partitioning.

In order to avoid over-counting, we need to find how many of p_{ij}^k's need to be counted. We define *border* portions of p_{ij}^k as the ordered tuple of (v_{m1}, v_{m2}) pairs, a subset of p_{ij}^k, where $(v_{m1}, v_{m2}) \in C$. We define $Borders$ as:

$$\begin{aligned} Borders(p_{ij}^k) &= ((v_{m1}, v_{m2}), (v_{m3}, v_{m4}), \dots) \\ &= \{(v_m, v_n) \mid (v_m, v_n) \subseteq p_{ij}^k \wedge (v_m, v_n) \in C\} \end{aligned} \tag{3}$$

Each border element (v_m, v_n) consists of an *exit* point (v_m) and an *entry* point (v_n). We define entry points (EP) as:

$$\begin{aligned} &EP(Borders(p_{ij}^k)) \\ &\quad = ((v_{m2}), (v_{m4}), \dots) \\ &\quad = \{(v_n) \mid \exists\ v_m\ s.t.(v_m, v_n) \subseteq p_{ij}^k \wedge (v_m, v_n) \in C\} \end{aligned} \tag{4}$$

The main part of a border tuple impacting the result, is the entry point to the next sub-graph. We define two paths are related by R if and only if they have border nodes with same entry points (with same order):

$$p_{ij}^{k1}\, R\, p_{ij}^{k2} \iff EP(Borders(p_{ij}^{k1}))\text{=}EP(Borders(p_{ij}^{k2})) \tag{5}$$

Relation R is an *equivalence* relation: it is *reflexive* $(p_1\ R\ p_1)$, *symmetric* $(p_1\ R\ p_2 \implies p_2\ R\ p_1)$ and *transitive* $(p_1\ R\ p_2 \wedge p_2\ R\ p_3 \implies p_1\ R\ p_3)$. Thus, R divides P_{ij} into disjoint sets. All paths inside the same equivalence class are *similar paths* and together, they carry the load from v_i to v_j only once. Thus, the duplication factor n_{ij} is defined as the number of *dissimilar paths* from v_i to v_j, which is the cardinality of the equivalence class:

$$n_{ij} = |P_{ij}/R| \tag{6}$$

Using the definition of duplication factor in Eq. 6, the multicast workload of a vertex u will be:

$$MW(u) = \sum_{v \in V} n_{ij}.IW(v) \tag{7}$$

Outgoing Unicast Load (UW). Outgoing Unicast Load (or unicast workload) of a name denotes the number of messages that leave one RP, and enter the other RP, because of that name. If a node v_k has no edge towards the other RP, its UW would be 0. Otherwise, it would be related to the number of its outgoing *effective edges* for every $IW(v_i)$ it receives:

$$UW^{v_i}(v_k) = n_{ik} \times ee_k^{v_i} \times IW(v_i) \tag{8}$$

where $UW^{v_i}(v_k)$ is the additional unicast load of v_k that is caused by incoming unicast load at v_i. The number of effective

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

edges out of v_k because of v_i is the number of edges that carry data from v_k to the other RP, *e.g.*, to a node v_j in the other RP. If v_j has more than one edge coming to it from the other RP, only one of them will be used, due to DFS's single-visit traversal. The total outgoing unicast load of v_k would be:

$$UW(v_k) = \sum_{\forall v_i \in V, v_i \neq v_k} UW^{v_i}(v_k) \tag{9}$$

3) Algorithm: While we can design an algorithm that strictly follows the formulas in §V-A2 for weight calculation, we can design more efficient algorithms that consume less memory. The mathematical representation of §V-A2 is used to prove and cross-validate the algorithm, as an alternative way of arriving at the final result. The calculation of these weights are done through an iterative diffusion algorithm which follows the propagation logic described in §IV-C. Algorithm 1 calculates the two weight values of each vertex, namely MW and UW (stored in maps 'multicastLoad' and 'unicastLoad' respectively) using propagation from each source vertex (that has incoming unicast load 'iLoad') to any reachable ancestor vertex, be it in the same or different sub-graph (part). Starting from the vertex 'nodeName', the algorithm traverses the graph and finds all the traversed vertices, using a modified DFS algorithm. Final weights of any traversed vertex within the same sub-graph as 'nodeName' will be incremented accordingly (by 'iLoad'). For any traversed vertex 'u' that is not in the same sub-graph as 'nodeName', *i.e.*, pointing to another RP, the procedure is recursively called, starting propagation from 'u', with 'iLoad' in the other sub-graph. Any vertex 'pu' having a link to another sub-graph will have its UW updated accordingly. 'modifiedDFS' (details omitted for brevity) performs traversal across two connected sub-graphs G1 and G2 according to our protocol. Its first and second input arguments represent the current and original DFS roots (to prevent cycles).

Algorithm 1 Weight Calculation in the Graph

```
procedure CALCULATE(Node nodeName, int iLoad, Graph G, Graph G0, Graph G1, Map <Node, Boolean> done)
▷ multicastLoad<Node, int> and unicastLoad <Node, int> are maps that store $MW$ and $UW$ values for each vertex (node)
    if nodeName in G0 then
        thisG←G0, otherG←G1
    else
        thisG←G1, otherG←G0
    for all Node u in thisG.modifiedDFS(nodeName, nodeName) do
        if u in thisG then
            thisG.multicastLoad.put(u, thisG.multicastLoad.get(u)+iLoad)
            G.multicastLoad.put(u, G.multicastLoad.get(u)+iLoad)
        else
            initialize Boolean todo←TRUE, Node pu ← “”          ▷ empty string
            initialize Map <Node, Boolean> done2 to all <v, FALSE>
            for all pu in thisG.adj.keyset do
                if u in thisG.adj.get(pu) AND
                  pu in thisG.modifiedDFS(nodeName,nodeName) then
                    if u NOT in thisG then
                        pu1 ← pu
                        if done2.get(pu)=TRUE then
                            todo←FALSE
            if todo=TRUE AND pu1 $\neq$ “” then
                done2.put(pu1, TRUE)
                thisG.unicastLoad.put(pu1, thisG.unicastLoad.get(pu1)+iLoad)
                G.unicastLoad.put(pu1, G.unicastLoad.get(pu1)+iLoad)
                CALCULATE(u, iLoad, G, G0, G1, done2)
```

4) Graph Partitioning Procedure: To prepare the graph for partitioning, the RP labels its local namespace sub-graph, which mainly consists of calculating and assigning appropriate weights explained earlier. The weights are calculated for each solution instance, including an initial solution provided by METIS [12]. We use Tabu search for iterative refinement of our graph partitioning solutions [14], [29] (algorithm details in [36]). Each solution (candidate) of the procedure provides a cut, which partitions the RP's namespace sub-graph into two *segments* (assuming bi-partitioning).

Initial solution: Tabu search typically starts with a random initial solution and improves it. To get a better initial partition [11], we try to use the result from the problem closest to ours – the (static) multi-criteria graph partitioning where the weights will not change according to the partition decisions. We use METIS for this stage as it is a highly popular tool that has been shown to be fast, while providing high quality solutions.

Objective Function: As mentioned, to reduce the search space, we adopt a bi-partitioning approach, where the heavily loaded RP's namespace is partitioned to be split between two RPs, *i.e.*, the current RP and the new RP. The objective (fitness) function we use to evaluate our partitioning solution, takes into account the cost of both segments (sub-namespaces managed by the two RPs) and provides a combined measurement of 'minimizing the imbalance between the two RPs', 'minimizing the maximum single segment load', and 'minimizing the inter-RP communication' (G_1 and G_2 are the two segments, associated with the two RPs):

$$\begin{aligned} F(G_1, G_2) = {} & \alpha \cdot |TC(G_1) - TC(G_2)| \\ & + \beta \cdot max(TC(G_1), TC(G_2)) + \gamma \cdot (UC(G_1) \\ & + UC(G_2)) \end{aligned} \tag{10}$$

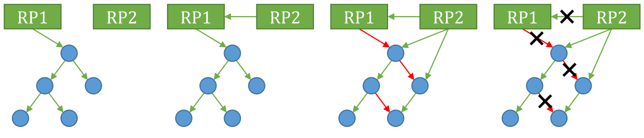
where α, β, and γ are optimization coefficients. Setting higher coefficients for some of the terms would result in the final solution being impacted more by those terms. However, the coefficients can be adjusted. We set all of them to 1 in our test cases, since these values appeared to provide reasonably good benefit, in our experiments. The aim is to minimize F. Function $UC(G_i)$ (segment total unicast cost) is the sum of cut edge weights initiated in G_i, and $MC(G_i)$ (segment total multicast cost) is the sum of all vertex weights in G_i. Furthermore, total load cost of a segment would be:

$$TC(G_i) = UC(G_i) + MC(G_i) \tag{11}$$

Stopping criterion: We allow both fixed and adaptive stop criteria. If fixed, a parameter *Max Iterations* i is pre-defined, and Tabu search stops when i is reached. Our adaptive stopping criterion, on the other hand, starts with an *Iteration Base* b, and any time the 'so far found best solution' is changed, b gets added to the current iteration number and makes up the new final iteration number. This ensures that our Tabu search procedure stops only after running with b iterations of no improvement. To prevent the Tabu procedure to keep iterating indefinitely, with this adaptive stopping criterion, an upper bound on the number of iterations is also specified.

B. Migrating Cores

Once the graph partitioning is done, the names in one segment need to be migrated to another core. The RP selection function is similar to that in IP multicast [35]. It may be performed by a network manager or calculated by a Network Coordinate function such as [45]. Once the RP is selected,



(a) Before (b) Init (c) After M1 (d) After M2

Fig. 8. Reliable RP splitting: RP1 relinquishing a name to RP2.

the process essentially migrates the names in the partitioned subspace to the other RP. However, this has to be done carefully because if a router discards the original subscriptions before it receives all the publications that are in-flight (before the original 'pipe' is drained), these publications will be lost.

To address this, we propose a 3-stage (make-before-break) solution to ensure reliable delivery during migration, as shown in Fig. 8. Before migration, we assume there is a multicast tree rooted at RP1 (Fig. 8(a)). When RP1 decides to move a name to RP2, in stage 1 (Fig. 8(b)), it notifies RP2 *and* also subscribes to RP2 (creating new green line). Meanwhile, it notifies the network that RP2 is now serving that name (routing update in IP, FIB propagation in NDN, or a GNRS update in MobilityFirst). RP2 now becomes the RP for the name, and routers with the new RP information will send publications to RP2. However, reusing the original multicast tree, we continue to make sure that the publications are delivered during the transient phase. Routers may have not yet updated the name-to-RP mapping, and there can be publications in-flight during the mapping update. Thus, some publications to those names will still reach RP1. We adopt the late-binding concept of MobilityFirst: when an RP receives a publication that is not served by itself. It hands the publication back to the network to then be forwarded to the correct RP accordingly.

At stage 2 (the 'make' stage), RP1 sends out a special marker packet (we call it M1) to all the nodes in the subscription tree. M1 is treated just as a normal multicast packet. To make sure that all the subscribers in the tree receive the M1 marker packet, RP1 has to send that packet after it is sure that the new mapping has propagated into the network and the subscriptions based on the old mapping have joined the tree. On receiving M1, routers subscribe towards the new RP and mark the original ones as 'stale' if the original entry in the subscription table is different from the new entry. Fig. 8(c) shows the subscription after M1 is propagated to the network. The green arrows are the new subscriptions and red arrows are the 'stale' ones. While we mark the subscriptions as stale, we do not delete them. When RP2 sends publications, it sends them along all the subscription links, to ensure delivery. A nonce can be used in the packets to eliminate redundant traffic during this transient phase.

After all the nodes subscribe to the new RP, RP1 can send a second marker packet (we call it M2) to start the third and final stage (the 'break' stage). On receiving this marker packet, the intermediate nodes clean up the 'stale' subscriptions (as is shown in Fig. 8(d)). When a node has no downstream subscriptions (*e.g.*, RP1 in the Fig.), it will unsubscribe from the upstream naturally. Since all the nodes have subscribed to the new RP, the M2 marker packet acts as the last packet in the pipe. Thus, we will not lose packets if we close the 'pipe' (unsubscribe) after we receive M2. We also use this mechanism to provide resiliency to RP failures [36].

TABLE I

SOLUTION QUALITY OF ALTERNATIVES AND GLOBAL OPTIMUM

Vertices	Edges	Optimum	METIS	POISE	Solution itr	Final itr
10	14	1,916	2,093	1,916	12	22
10	20	2,434	2,736	2,434	14	24
15	19	1,400	1,763	1,400	6	21
15	21	4,744	5,876	4,744	5	20
15	26	6,753	10,460	6,753	28	43
15	65	6,119	16,271	6,119	6	21
20	29	2,594	3,162	2,856	20	40
20	42	9,342	18,905	9,342	10	30
20	89	7,689	15,587	10,480	10	30
25	44	5,966	7,230	5,966	27	52

VI. EVALUATION

To evaluate POISE, we compare it to a number of existing and theoretical alternatives. In terms of overall architectures, we compare POISE to NDN/CNS [5], a recipient-based push-based pub/sub architecture for notification systems, which is the closest architecture to ours. For namespaces, we compare POISE's graph-based naming with the most advanced state-of-the-art ICN naming, which is NDN's hierarchical naming (as in CNS as well). For load-splitting, we compare POISE to the most popular graph partitioning tool METIS [12]. We use the same design principles of the simulator in [5], while adding the functionality of our information layer graph namespace design and splitting procedures. Our simulator is open-sourced and available in [46]. For the partitioning component, we use the current implementation of METIS (and ParMETIS [13]), plus our refinement and weight/objective calculation procedures. We also describe our POISE RP implementation on DPDK, and provide micro-benchmarking results to justify POISE.

A. Evaluating the Graph Partitioning Algorithm

In this section, we evaluate the quality of POISE's graph partitioning, *i.e.*, the hybrid "METIS+Tabu" algorithm. To compare the quality of solutions provided by METIS, and METIS+Tabu (starting from METIS and then doing a Tabu search) with the global optimum (using exhaustive search), we use 10 relatively small graphs, randomly picked and labelled with weights (taken from [47]), as described in Table I. METIS+Tabu uses v iterations with Tabu tenure of $\sqrt{v}$ for a graph with v vertices. For finding the global optimum (*i.e.*, *the* optimal solution), we generated all possible solutions using a brute-force (exhaustive search) approach. For the METIS+Tabu (POISE) case, Table I also shows the Tabu iteration at which the best solution was found ('Solution itr') as well as the number of the last iteration ('Final itr'). For the cases in Table I, METIS+Tabu (the approach in POISE) finds the optimal solution most of the time (*e.g.*, in 8 out of the 10 cases considered). It also reaches the global optimum within a reasonable number of iterations. Comparing the complexity of the Tabu search and the brute-force approaches, we see a significant benefit of using our Tabu search approach. While the brute-force approach finds the global optimum by checking $2^{n-1} - 2$ solutions (assuming

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

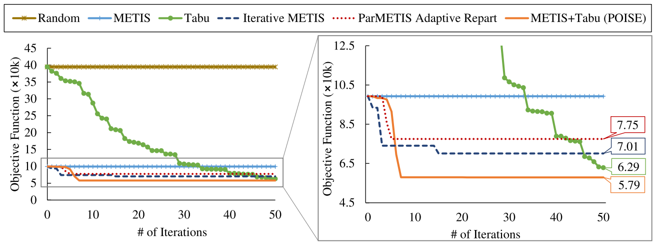


Fig. 9. Effectiveness of different graph partitioning approaches.

bi-partitioning and filtering out of duplicate permutations and no-cut solutions), Tabu search finds that solution or one reasonably close to it by checking $O(iv)$ candidate solutions, which in our case is $O(v^2)$, since we set the number of iterations i to be $O(v)$ and at each iteration, $O(v)$ neighboring solutions are visited and evaluated. Even though in Table I, POISE found the exact global optimum for most of the cases, this does not necessarily have be to the case for all input graphs. In particular, for two graphs, namely $G = (20, 29)$ and $G = (20, 89)$, POISE did not reach the global optimum. For these two cases, increasing the Tabu iterations by a factor of 10 did not improve the solution either. However, as Table I shows, POISE's meta-heuristic guided search-based partitioning does find the global optimum for a majority of times, and reaches a near-optimal (at least nearer compared to METIS) solution in all cases. Additionally, as the table also shows, POISE consistently achieves better solutions, *i.e.*, closer to the optimum, compared to the state-of-the-art METIS, further showing the benefit of POISE's partitioning.

Finding the global optimum through brute-force search is not computationally feasible for large graphs, as the number of candidate solutions to visit grows rapidly exponentially. For larger graphs, we only need to do a comparative evaluation, showing that METIS+Tabu finds relatively better, and in most cases, significantly better solutions, than alternative approaches. To show this comparison, we use one of the graphs available online in the repository in [47] (from its "AT&T graphs" package). It is a directed graph with 50 vertices & 84 edges. The graph is unweighted, so we assign random values between 0 and 100 to each vertex, to denote the incoming unicast load for each name. Fig. 9 shows the comparison across different alternatives, in terms of the quality of solution (objective function) for this graph. The following scenarios are used: Random (average of three randomly generated solutions), METIS, Tabu (average of three runs of Tabu-only starting from random initial solution), Iterative METIS, ParMETIS with Adaptive Repartitioning (parallelized on two processors, with the 'coupling' of sub-graphs with processors for the best performance [13]), and METIS+Tabu (POISE). Note that the "Random" and "METIS" scenarios are not iterative procedures therefore achieve a fixed solution quality. We vary the number of iterations for the solutions that support refinement (*i.e.*, dynamic solutions), with a fixed stop criterion, to show how quickly the search-based approaches converge to a good quality solution.

Fig. 9 shows that METIS+Tabu outperforms the rest. Using METIS as initial solution (METIS+Tabu) *vs.* starting from a random initial point (Tabu) is very effective as the algorithm reaches its convergence point (for the range of iterations we examined) much faster (with fewer iterations). The Tabu-only method outperforms METIS, Iterative METIS, and ParMETIS Adaptive Repart only after a relatively large (above 40) number of iterations. The METIS+Tabu approach outperforms METIS very early, after just 5 iterations. It also outperforms Iterative METIS and ParMETIS Adaptive Repartitioning early, before the 10^{th} iteration. This shows that the Tabu search is the preferred refinement approach. The random partitioning solution is much worse than the other alternatives. Fig. 9 also shows the importance of choosing an appropriate stop criterion. The number of iterations being too small precludes reaching a good solution, and it being excessively large results in waste of time and compute resources.

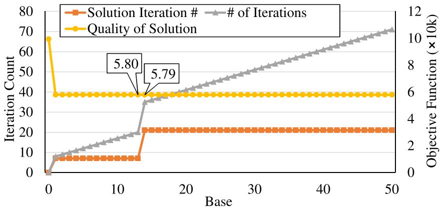


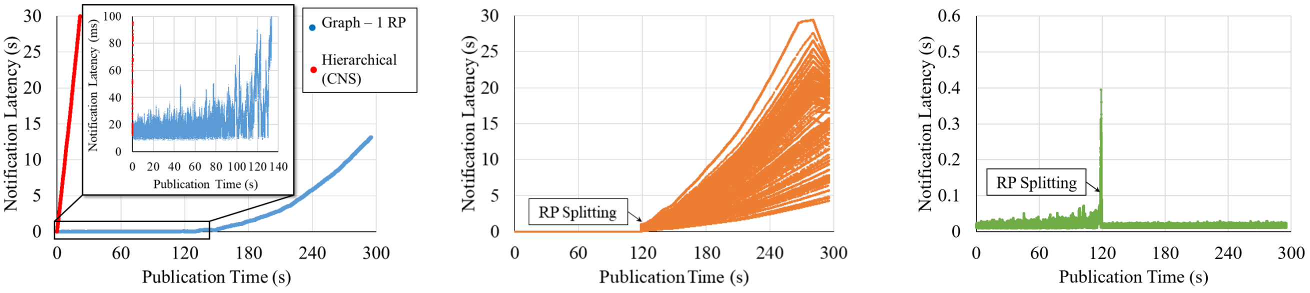
Fig. 10. Impact of base in adaptive stop criteria.

We also examine using an adaptive stop criterion. Fig. 10 shows the impact of the base parameter (§V-A) on the quality of solution found by our graph partitioning (yellow line) in terms of objective function (right-side y-axis). This shows that the base parameter in an adaptive stop criterion needs to be selected carefully as well; *i.e.*, not too small or too large. The jump between base values of 13 and 14 indicates that a new solution is found with base of 14 that would not have been found with smaller base values, showing an example of how Tabu search escapes local optima. Fig. 10 also shows the resulting number of iterations (gray line) and the iteration where the last improvement was found (red line), for each base value, in terms of count/number (left-side y-axis). The difference between these two last numbers shows the number of wasted iterations for that setting of base value. In this example, we see that with a base of higher than 15, increasingly more and more iterations are wasted. Results on more graphs are provided in [36], which further show the improved quality of solution in POISE.

B. Overall Solution Evaluation

To evaluate the performance of POISE, we implemented an event-driven, packet-level simulator. The simulator supports name-based pub/sub, exploring different alternatives within that paradigm, using any type of multicast network layer underneath. We can compare name-based multicast to alternatives such as pull-based pub/sub, IP multicast-based pub/sub, and broadcast-based pub/sub such as those examined in [3]. To evaluate the behavior, we needed a realistic network environment with a number of forwarding routers and end-points that are publishers and subscribers. For this, the network topology we use to evaluate POISE and compare with various alternatives is the Rocketfuel 1221 Telstra [48] with some modification for a state-wide disaster scenario. Our topology contains 46 core routers, with additional 231 routers

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.



(a) Hierarchical & Graph - 1RP

(b) Graph - Random Split

(c) POISE

Fig. 11. Notification latency over time in different solutions (Note the difference in the scale of notification latency in POISE).

placed at the edge each linking to 2 core routers closest to them. We use the graph-based "Disaster Management" category namespace from the Wikipedia database as our namespace [49]. Exploring 6 levels below that category, we obtain 489 categories and 732 relationships. If we seek to extract a set of hierarchies based on the approach in [9], we obtain 1,468 hierarchical names. We use the associated pages and files from the Wikipedia database (8,577 items total, 436 per category maximum, 17.49 on average per category) as the publications. We duplicated each content 60 times (514,620 publications) and ordered their publication randomly to load the network. While the namespace is static in our experiments, the publication workloads are dynamic and vary. Publications are generated using a Poisson distribution (to model human behaviors such as calling for, or offering to, help) with a *monotonically increasing arrival rate* over time (to model the increasing nature of such publications, as the disaster unfolds and more people become aware and get involved). We experiment with two example publication workloads: 1) moderate (average arrival rate varying from 1,500 pkt/s to 2,000 pkt/s) and 2) intense (arrival rate varying from 1,500 pkt/s to 3,500 pkt/s). We create 6 subscribers for each category (2,934 in total), distributed randomly on the 231 edge routers. Eventually we generate 20,022,480 delivery events.

Experiments with moderate workload: We first consider the notification latency, to deliver a publication to all recipients. This reflects the impact of queuing in the network that arises from having to route through an RP, the selection of an appropriate number of RPs at the correct point in the network topology, adapting to the namespace and workload. We also look at the total network traffic to understand scalability.

We compare the performance of POISE with a number of alternatives. First, is the use of a strict hierarchical namespace (as in NDN/CNS). To be liberal to the hierarchical alternative, we avoid each subscriber having to subscribe to every name. Therefore, when there are multiple hierarchical names for a category, he subscribes to *any* one of the names. The publisher publishes to *all* the hierarchical names of the category. We also compare with having a single RP (no splitting), as well as a simple random splitting of the RP to one of the nodes in the network. The latter is used to demonstrate the need to use a near-optimal splitting of the RPs and load balancing.

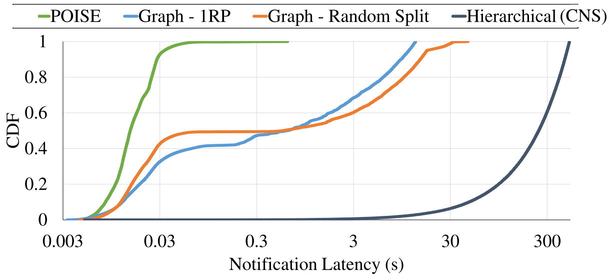


Fig. 12. Notification latency CDF in different solutions.

TABLE II
AVERAGE NOTIFICATION LATENCY & AGGREGATE NETWORK TRAFFIC

Solution	Notification Latency (s)	Network Traffic (Gb)
POISE	0.018	492.39
Graph - 1RP	2.741	483.08
Graph - Random Split	4.725	625.69
Hierarchical (CNS)	247.742	866.27

From the CDF of the notification latency in Fig. 12 (and the average reported in Table II), due to the high workload on the RP caused by hierarchical names, the notification latency is excessive. Having only 1 RP (graph-1RP) as well as random splitting of RPs perform reasonably at lower loads (for rates <1700 pkt/s) and are even better than using hierarchical names at low loads. However, at higher workloads, random split and hierarchical names perform poorly compared to POISE as well as even having just one RP.

Fig. 11 shows the notification latency as the load gradually increases, for all the solutions. With a random split, Fig. 11(b)), the notification latency goes up very rapidly after the split, because the entire system is overloaded by packets sent back and forth between RPs. It is even worse than having a single RP, with no splitting (blue line in Fig. 11(a)). This shows the importance of a sensible partitioning algorithm. For the same workload, the latency of POISE (with METIS+Tabu, Fig. 11(c)) is dramatically better (by 2-orders of magnitude). As the load goes up, RP partitioning is triggered. For a short transient period, queuing causes a relatively small (compared to other alternatives) increase in latency. But congestion is immediately relieved by RP splitting and the latency drops back down. The maximum transient latency is 400 ms with POISE, compared to multiple seconds with other alternatives.

Next, we look at the total network traffic, summarized in Table II. Splitting the RP in POISE results in slightly higher traffic (∼1%) due to the unicast between RPs, compared to having only a single RP. Yet by doing so, we avoid the significant latency impact of congestion. Random splitting of

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE III

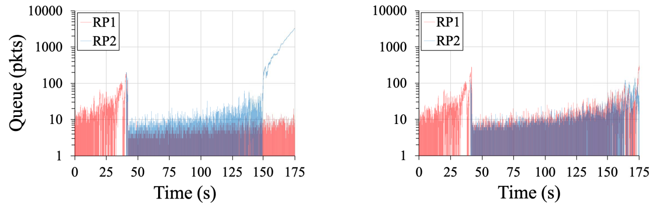
COMPARISON OF METIS AND POISE'S PARTITIONING

Metric	METIS	POISE
Moderate workload		
Average latency (s)	0.018171	0.018147
Aggregated traffic (Gb)	491.242	492.392
Max Load - 2RP (#msgs)	1,321,220	1,230,533
Load imbalance - 2RP (#msgs)	360,693	633
Inter-RP messages - 2RP (#msgs)	136,571	314,139
Objective - 2RP	1,818,484	1,545,305
Intense workload		
First split time (s)	40.868	
Second split time (s)	150.388	174.252
Average latency in [0,170s] (s)	0.049686	0.020387

the RP performs much worse (*and* causing 27% more traffic compared to sensible splitting). In fact, if one were to just consider the relative increase in the amount of traffic because of RP splitting (compared to having just one RP and not having any RP splitting), then random splitting with 133.3 Gb of extra traffic results in 14.3 times more than POISE (9.3 Gb) in terms of extra traffic. Compared to the hierarchical solution, the graph-based solution of POISE reduces the amount of network traffic dramatically (by 75.9%) since we do not have to deal with the extra names and publications.

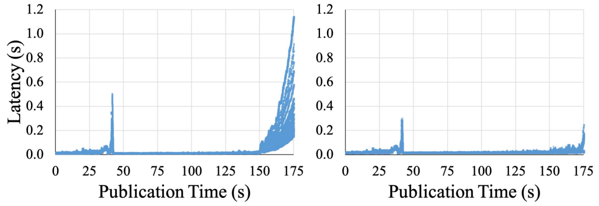
To dig a little deeper into the impact of the partitioning method used, we provide more detailed metrics in Table III to compare the use of METIS and METIS+Tabu (POISE). For the moderate input workload, using METIS+Tabu leads to slightly (24μs) improved average notification latency (per delivery), while adding 0.002% total traffic. The reason for this better latency is better balance, and thus less queuing delay, even at the cost of slightly more traffic (just like a single RP having the least total traffic in Table II). The load metrics (in terms of # of messages) measure the RP load from the time of the split until the end of simulation (*i.e.* during the time the system has 2 RPs; labeled with '$-$2RP'). The table shows the values for the three terms in Eq.10. For METIS+Tabu, maximum RP load and load imbalance are significantly better, while for METIS, the # inter-RP messages is lower (leading to slightly less traffic). These combined, make METIS+Tabu's solution more balanced with a lower peak, as confirmed by the 'Objective' (sum of the above three terms), validating the effectiveness of our partitioning approach.

Experiments with intense workload: The benefit of METIS+Tabu over METIS is even more significant when we generate a higher intensity workload. The same publication trace (for $\sim$300s) was generated over a shorter duration ($\sim$217s) by increasing the average inter-arrival rate. We also increase the RP splitting threshold. Table III shows the time at which the first and the second RP splits occur; the first split is same for both (*i.e.*, 40.868s) while the second split occurs $\sim$24s later with METIS+Tabu compared to METIS (174s *vs.* 150s). The better balance with METIS+Tabu helps the single RP maintain the namespace for a longer time with lower dissemination latency; the same RP is used for 21% longer than the case of METIS. This is important, since the splitting procedure introduces protocol overhead (§V-B), with additional notification latency for a short period, as shown in Fig. 11(c). Therefore, postponing splitting and reducing its frequency during the lifetime of the overall system is beneficial. However, if the split is postponed for too long, this latency would increase significantly, as seen in Table III. For the period of [0,170s], the average notification latency of METIS is more than twice the latency of METIS+Tabu. Fig. 13 shows the instantaneous queue size of each RP in the two cases, for the period of [0,175s]. Most importantly, it shows the better balance between the two RPs in case of POISE (METIS+Tabu, Fig. 13(b)) than METIS (Fig. 13(a)). We also see that the size of the queue in RP2 for METIS goes up above 3,000 during that period, much larger than METIS+Tabu, which only goes up to 140 for the same time. As the figure shows, RP1's queue size is a little higher in POISE than METIS. However, the queue grows much more at RP2 in METIS than with POISE. This is the tradeoff that POISE makes, producing a better balance between the two RPs, thus helping prolong the need for splitting the RPs. The latency per publication for the intense workload is also shown in Fig. 14, indicating a much higher increase for METIS (Fig. 14(a), seeing congestion after 150s) compared to METIS+Tabu (POISE, Fig. 14(b), which stays low throughout, until 175s).



(a) METIS (b) METIS+Tabu (POISE)

Fig. 13. RP queue sizes for intense workload.



(a) METIS (b) METIS+Tabu (POISE)

Fig. 14. Notification latency for intense workload.

C. Implementation

We implement POISE in C to demonstrate the feasibility and efficiency of the protocol. To eliminate the performance impact of handling kernel interrupts, we take advantage of Data Plane Development Kit (DPDK) [50] poll-mode driver so that our user-space program can receive/send data directly from/to the NIC. We implement a routing table, a subscription table and a neighbor table (similar to ARP tables) on every forwarding engine to route packets to the RP and multicast data to the subscribers (see Fig. 15). On the RPs, we also add logic for expansion based on the graph-based namespace. The graph is implemented using a hash table whose key is the parent name and the value is a list of descendant names. A breadth-first search (BFS) is performed for each packet reaching the RP. To maximize throughput, we use lock-free data structures [51] and use read-copy-update (RCU) [52] on

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

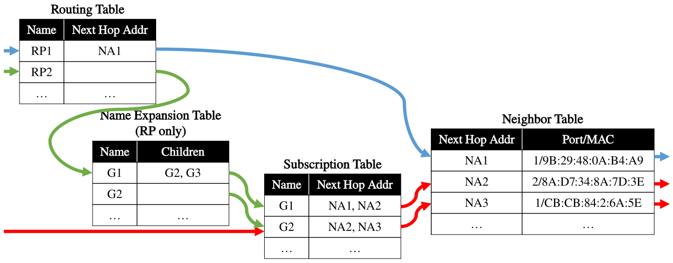


Fig. 15. Data structures and data flows in the POISE implementation. Blue: upstream publication packet with destination=RP1; Red: downstream publication with name=G2; Green: publication with name=G1 expanded at the RP (flow after subscription table omitted).

the data entries. To further reduce the latency for multiple hash calculations, we adopt the technique of "chasing pointers". For example, in the routing table in Fig. 15, the value of name `RP1` is `NA1`. When a data packet with destination=`RP1` reaches the forwarding engine, the forwarding engine has to perform a hash lookup for `RP1` in the routing table and another hash lookup for `NA1` in the neighbor table to determine the outgoing interface and the MAC address to encapsulate the packet (the blue flow in the figure). This involves 2 hash lookups. In our implementation, instead of writing the value of `NA1` in the routing table, we have a pointer pointing to the neighbor table entry of `NA1`. When the data packet reaches the forwarding engine, we only need to perform 1 lookup (in the routing table) and follow the pointer to get the outgoing port and MAC address. We thus save the second hash calculation and table lookup. This technique reduces the overhead dramatically on the RPs where multiple lookups have to be performed for the name expansion table in BFS.

We perform micro-benchmarks on the implementation using servers in the ORBIT testbed [53]. The machines use Intel Xeon E5-2640 CPU @2.4GHz (20-cores, hyper-threading turned off) with 256GB of memory, and a Mellanox MT27710 25 Gbps NIC that supports DPDK. We schedule our program on cores 10-19 to prevent cross-NUMA node accesses. We are able to forward ∼15.3 million packets per second (Mpps) when dealing with upstream packets (of 64 Bytes in size). For downstream packets, we are able to achieve similar performance when there is only a single next-hop downstream. When there are additional next-hops downstream, the performance drops dramatically (down to serving ∼4.1Mpps incoming packets for 2 next-hops, and ∼3.2Mpps incoming packet for 3 next-hops). This is mainly due to the overhead of DPDK copying packets or the use of multi-segment packets depending on the replication solution we choose. In comparison, the overhead for name expansion, which we observe on the RP module is much higher. Performing the name expansion on a random graph with 128 names, the processing rate only achieves ∼0.8Mpps, even with only 1 next-hop. This demonstrates the necessity of limiting the namespace expansion only to the information layer (otherwise, every single forwarding engine has to behave like the RP module and suffer the performance penalty). Then, with the approach such as POISE, we can have multiple RPs in the network to share the load, as needed, dynamically.

VII. Conclusion

We proposed POISE, an architecture for recipient-based pub/sub for disaster management, supporting free-form graph-based namespaces and automatic load splitting to eliminate traffic concentration based on a novel hybrid graph partitioning algorithm. Our simulation and micro-benchmarking results show that POISE is efficient and scalable, compared to alternatives: its graph-based namespace outperforms the state-of-the-art hierarchical namespace of NDN [6]; its overall network architecture extends the recipient-based pub/sub framework of CNS [5]; its partitioning outperforms the popular graph partitioner METIS/ParMETIS [12]; POISE's RP-based name expansion is expected to outperform expansion at every router.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2009, pp. 1–12.

[2] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From PSIRP to PURSUIT," in *Proc. Int. Conf. Broadband Commun., Netw. Syst.*, 2010, pp. 1–13.

[3] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. K. Ramakrishnan, "COPSS: An efficient content oriented publish/subscribe system," in *Proc. ACM/IEEE 7th Symp. Archit. Netw. Commun. Syst.*, Oct. 2011, pp. 99–110.

[4] M. Yuksel *et al.*, "Cross-layer failure restoration of IP multicast with applications to IPTV," *Comput. Netw.*, vol. 55, no. 9, pp. 2329–2351, Jun. 2011.

[5] J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "CNS: Content-oriented notification service for managing disasters," in *Proc. 3rd ACM Conf. Inf.-Centric Netw.*, Sep. 2016, pp. 122–131.

[6] L. Zhang *et al.*, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[7] A. Venkataramani, J. Kurose, D. Raychaudhuri, K. Nagaraja, M. Mao, and S. Banerjee, "MobilityFirst: A mobility-centric and trustworthy internet architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 74–80, Jul. 2014.

[8] A. Afanasyev *et al.*, "NFD developer's guide," NDN, Shanghai, China, Tech. Rep. NDN-0021, 2018.

[9] S. S. Adhatarao, J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "Comparison of naming schema in ICN," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jun. 2016, pp. 1–6.

[10] J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "G-COPSS: A content centric communication infrastructure for gaming applications," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst.*, Jun. 2012, pp. 355–365.

[11] A. Pınar and B. Hendrickson, "Partitioning for complex objectives," in *Proc. 15th Int. Parallel Distrib. Process. Symp. (CDROM)*, Washington, DC, USA, 2001, pp. 1–7.

[12] G. Karypis. (2013). METIS—Serial graph partitioning and fill-reducing matrix ordering, version 5.1.0. University of Minnesota, Minneapolis, MN, USA. [Online]. Available: http://www.cs.umn.edu/~metis

[13] G. Karypis and K. Schloegel. (2013). *ParMETIS—Parallel Graph Partitioning and Fill-Reducing Matrix Ordering, Version 4.0*. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

[14] Wikipedia. (2018). *Tabu Search*. [Online]. Available: https://en.wikipedia.org/wiki/Tabu_search

[15] J. Seedorf *et al.*, "The benefit of information centric networking for enabling communications in disaster scenarios," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2015, pp. 1–7.

[16] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content based routing with Elvin4," in *Proc. AUUGk*, 2000, pp. 1–11.

[17] Y. Diao, S. Rizvi, and M. J. Franklin, "Towards an internet-scale XML dissemination service," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2004, pp. 612–623.

[18] S. Mukherjee, F. Bronzino, S. Srinivasan, J. Chen, and D. Raychaudhuri, "Achieving scalable push multicast services using global name resolution," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[19] J. Chen, M. Jahanian, and K. K. Ramakrishnan, "Black ice! Using information centric networks for timely vehicular safety information dissemination," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jun. 2017, pp. 1–6.

[20] (2019). *Wikipedia: Outline of Knowledge*. [Online]. Available: https://en.wikipedia.org/wiki/Portal:Contents/Outlines

[21] R. Angles and C. Gutierrez, "Querying RDF data from a graph database perspective," in *Proc. Eur. Semantic Web Conf.*, 2005, pp. 346–360.

[22] (2019). *Kubernetes*. [Online]. Available: https://kubernetes.io/

[23] D. Di Sarli and F. Geraci, "GFS: A graph-based file system enhanced with semantic features," in *Proc. Int. Conf. Inf. Syst. Data Mining (ICISDM)*, 2017, pp. 51–55.

[24] A. E. Feldmann and L. Foschini, "Balanced partitions of trees and applications," *Algorithmica*, vol. 71, no. 2, pp. 354–376, 2015.

[25] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Aug. 1999.

[26] A. Bhatele, S. Fourestier, H. Menon, L. V. Kale, and F. Pellegrini, "Applying graph partitioning methods in measurement-based dynamic load balancing," Lawrence Livermore Nat. Lab., Livermore, CA, USA, Tech. Rep. LLNL-TR-532851, 2012.

[27] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan, "Goldilocks: Adaptive resource provisioning in containerized data centers," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 666–677.

[28] I. Stanton and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2012, pp. 1222–1230.

[29] E. Rolland, H. Pirkul, and F. Glover, "Tabu search for graph partitioning," *Ann. Oper. Res.*, vol. 63, no. 2, pp. 209–232, Apr. 1996.

[30] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning," *Oper. Res.*, vol. 37, no. 6, pp. 865–892, Dec. 1989.

[31] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. Sci. Comput.*, vol. 25, no. 6, pp. 1837–1859, Jan. 2004.

[32] I. Moulitsas and G. Karypis, "Partitioning algorithms for simultaneously balancing iterative and direct methods," Dept. Comput. Sci., Minnesota Univ. Minneapolis, Minneapolis, MN, USA, Tech. Rep. 04-014, 2004.

[33] R. H. Bisseling and W. Meesen, "Communication balancing in parallel sparse matrix-vector multiplication," *Electron. Trans. Numer. Anal.*, vol. 21, no. 1, pp. 47–65, 2005.

[34] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, *Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised)*, document RFC 4601, Aug. 2006.

[35] Y.-D. Lin, N.-B. Hsu, and C.-J. Pan, "Extension of RP relocation to PIM-SM multicast routing," in *Proc. IEEE Int. Conf. Commun. Conf. Rec. (ICC)*, Jun. 2001, pp. 234–238.

[36] M. Jahanian, J. Chen, and K. K. Ramakrishnan, "Graph-based namespaces and load sharing for efficient information dissemination in disasters," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–12.

[37] A. Tagami *et al.*, "Name-based push/pull message dissemination for disaster message board," in *Proc. IEEE Int. Symp. Local Metrop. Area Netw. (LANMAN)*, Jun. 2016, pp. 1–6.

[38] H. M. A. Islam, D. Lagutin, A. Lukyanenko, A. Gurtov, and A. Ylä-Jääski, "CIDOR: Content distribution and retrieval in disaster networks for public protection," in *Proc. IEEE 13th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2017, pp. 324–333.

[39] E. Monticelli, B. M. Schubert, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "An information centric approach for communications in disaster situations," in *Proc. IEEE 20th Int. Workshop Local Metrop. Area Netw. (LANMAN)*, May 2014, pp. 1–6.

[40] C. Gündoğan, P. Kietzmann, T. C. Schmidt, and M. Wählisch, "Information-centric networking for the industrial Internet of Things," in *Wireless Networks and Industrial IoT*. Cham, Switzerland: Springer, 2021, pp. 171–189.

[41] M. Zhu *et al.*, "CCDN: Content-centric data center networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3537–3550, Dec. 2016.

[42] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.

[43] S. Shannigrahi, C. Fan, and C. Partridge, "What's in a name?: Naming big science data in named data networking," in *Proc. 7th ACM Conf. Inf.-Centric Netw.*, Sep. 2020, pp. 12–23.

[44] M. Jahanian and K. K. Ramakrishnan, "Name space analysis: Verification of named data network data planes," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 848–861, Apr. 2021.

[45] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 15–26, Aug. 2004.

[46] *Poise Simulator*. Accessed: Mar. 18, 2021. [Online]. Available: https://github.com/SAIDProtocol/NetworkSimulator

[47] GDdata. *Graph Drawing*. Accessed: May 21, 2018. [Online]. Available: http://www.graphdrawing.org/data.html

[48] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proc. 2nd ACM SIGCOMM Workshop Internet Measurment (IMW)*, 2002, pp. 231–236.

[49] Wikipedia. *Category: Disaster Management*. Accessed: May 21, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Category:Disaster_management

[50] *Data Plane Development Kit*. Accessed: Jun. 25, 2020. [Online]. Available: https://www.dpdk.org/

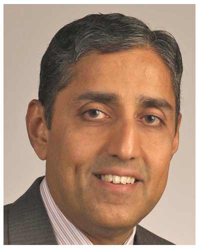
[51] *24. Hash Library—Data Plane Development Kit 20.08.0-RC2 Documentation*. Accessed: Jun. 25, 2020. [Online]. Available: https://doc.dpdk.org/guides/prog_guide/hash_lib.html

[52] *6. RCU Library—Data Plane Development Kit 20.08.0-RC2 Documentation*. Accessed: Jun. 25, 2020. [Online]. Available: https://doc.dpdk.org/guides/prog_guide/rcu_lib.html

[53] D. Raychaudhuri *et al.*, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *Proc. IEEE Wireless Commun. Netw. Conf.*, vol. 3, Mar. 2005, pp. 1664–1669.

Mohammad Jahanian received the B.S. degree from the University of Tehran in 2012 and the M.S. degree from the Sharif University of Technology in 2014. He is currently pursuing the Ph.D. degree with the University of California, Riverside. His current research interests include information-centric networks, formal verification, and distributed systems.

Jiachen Chen received the B.E. and M.E. degrees in software engineering from Fudan University, China, in 2007 and 2010, respectively, and the Ph.D. degree in computer science from the University of Göttingen in 2015. He is currently a Post-Doctoral Associate with the WINLAB, Rutgers University. His research interests include information-centric networks (ICN), the Internet of Things (IoT), cloud computing, and network management.



K. K. Ramakrishnan (Fellow, IEEE) received the M.Tech. degree from the Indian Institute of Science in 1978 and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, USA, in 1981 and 1983, respectively. He is currently a Professor of computer science and engineering with the University of California, Riverside. Previously, he was a Distinguished Member of Technical Staff with AT&T Labs-Research. Prior to 1994, he was a Technical Director and a Consulting Engineer in networking with Digital Equipment Corporation. From 2000 to 2002, he was a Founder and the Vice President with TeraOptic Networks, Inc. He has published over 300 papers and has 185 patents issued in his name. He is an ACM Fellow and an AT&T Fellow, recognized for his fundamental contributions on communication networks, including his work on congestion control, traffic management, and VPN services.