Software Managed Networks via Coarsening

Pradeep Dogga*
UCLA
Los Angeles, CA, USA

Rachee Singh*
Cornell University
Ithaca, NY, USA

Suman Nath
Microsoft Research
Redmond, WA, USA

Ravi Netravali
Princeton University
Princeton, NJ, USA

Jens Palsberg
UCLA
Los Angeles, CA, USA

George Varghese
UCLA
Los Angeles, CA, USA

ABSTRACT

We propose moving from Software Defined Networks (SDN) to Software Managed Networks (SMN) where *all* information for managing the life cycle of a network (from deployment to operations to upgrades), across all layers (from Layer 1 through 7) is stored in a central repository. Crucially, a SMN also has a generalized control plane that, unlike SDN, controls all aspects of the cloud including traffic management (e.g., capacity planning) and reliability (e.g., incident routing) at both short (minutes) and large (years) time scales. Just as SDN allows better routing, a SMN improves visibility and enables cross-layer optimizations for faster response to failures and better network planning and operations. Implemented naively, SMN for planetary scale networks requires orders of magnitude larger and more heterogeneous data (e.g., alerts, logs) than SDN. We address this using *coarsening* — mapping complex data to a more compact abstract representation that has approximately the same effect, and is more scalable, maintainable, and learnable. We show examples including Coarse Bandwidth Logs for capacity planning and Coarse Dependency Graphs for incident routing. Coarse Dependency Graphs improve an incident routing metric from 45% to 78% while for a distributed approach like Scouts the same metric was 22%. We end by discussing how to realize SMN, and suggest cross-layer optimizations and coarsenings for other operational and planning problems in networks.

CCS CONCEPTS

• **Networks** → **Network management**.

KEYWORDS

Network Management, AIOps, Capacity Planning

HotNets '25, November 17–18, 2025, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11
https://doi.org/10.1145/3772356.3772393

ACM Reference Format:
Pradeep Dogga*, Rachee Singh*, Suman Nath, Ravi Netravali, Jens Palsberg, and George Varghese. 2025. Software Managed Networks via Coarsening. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3772356.3772393

1 INTRODUCTION

Indeed, when people operate in silos, companies may miss innovation opportunities. – Peter Drucker

Planetary-scale networks including clouds like Amazon AWS, Google Cloud, and Microsoft Azure are complex to deploy. Managing these massive networks demands significant operational resources due to frequent production outages [15, 29, 37] and necessitates analysis of large amounts of heterogeneous operational data for critical tasks, including capacity planning [39] and engineering fault tolerance [16].

Problem: Siloed Network Management. The inherent complexity of these networks is exacerbated by what we call *siloed network management*—a practice where network management responsibilities are strictly partitioned by layers within the network stack. For example, in most hyperscalers, one team maintains the physical network infrastructure like optical fiber, another oversees Layer 2/3 operations like tunneling [42] and SDN-based traffic engineering (TE) [19, 20, 26, 27], and a third team owns operations of the transport layer. Each of these teams builds and maintains its part of the system largely in isolation. While SDN has made inroads within individual layers of the networking stack, the benefits of SDN have remained trapped inside layer-based silos.

Unfortunately, this compartmentalization has a cost as management experts have long realized [4]. Operational boundaries across layers, both in software and organizational structure, lead to missed dependencies between network configuration across the layers. This causes seemingly minor changes in one layer to cascade into major outages elsewhere. Consequently, network issues become challenging and time-consuming to diagnose and resolve. We highlight four

*These authors contributed equally to this work

real-world examples where this fractured model of network management has caused operational problems:

Capacity Planning and TE in the Dark: Engineering teams managing optical fiber augment capacity on inter-datacenter (DC) links of the wide-area network (WAN) to prevent bottlenecks. Independently, TE controllers at Layer 3 optimize traffic distribution across those links. But the relationship is bidirectional: TE decisions affect link utilization, influencing which links should be upgraded by the capacity teams, and link capacities inform how traffic is engineered. However, the tools and metrics all teams use are disconnected. This mismatch can lead to capacity teams upgrading links that TE has recently overloaded—regardless of whether the link can even be upgraded, say due to fiber constraints in the ground. The result is wasted planning cycles and misaligned optimization.

Wavelength Modulation and Resilience: Pushing optical wavelengths to higher data rates increases their susceptibility to failure [40]. At Layer 3, each wavelength maps to one or more logical inter-DC links [39, 48]; when a wavelength fails, the logical link drops, and the routing layer must reconverge [19, 20, 26, 27, 46]. These re-convergences are disruptive, especially when they occur frequently. Yet the teams tuning physical-layer parameters do so unaware of the higher-layer effects. In one case we observed, it took weeks for cloud operators to trace recurring routing flaps to an aggressive configuration at the optical layer.

WAN link flaps impacting cluster traffic: Link flaps in the public-WAN resulted in unsuccessful Pingmesh [17] probes originating from a DC cluster. The incident was first (wrongly) routed to the cluster team, causing resolution in hours because it was done manually by the cluster and WAN teams meeting.

Database service failure impacting downstream services: A partial failure in a database service caused alerts in six different services that depend on it. Each service, after alert triaging, independently created an incident and was assigned it a low priority (since the local impact was insignificant). Creation of six "unique" incidents caused redundant investigation, and the low priority caused delayed and manual investigation. Alerts from all services should have been triaged together to create a single high priority incident using the global view.

These examples point to a deeper problem: some classes of network problems are inherently cross-layer and cross-team, yet our current operational models resist cross-layer reasoning. Solving them demands holistic data collection and coordinated decision-making across layers. But today's architectures lack the fundamental capability to even observe such dependencies, let alone optimize for them.

We argue for a new approach—one where networks retain cross-layer observability and expose relevant state across traditional silos. With such relevant data, engineers could reason about inter-dependent phenomena and co-design mechanisms that span layers of networking stack. Such visibility is a prerequisite for realizing powerful cross-layer optimizations [24] and, more importantly, for empowering engineers to reason about the network as a unified system rather than a stack of opaque boxes. We believe breaking down these silos is not just a question of architecture—it is a foundational step toward enabling the next generation of cloud-scale networking innovation.

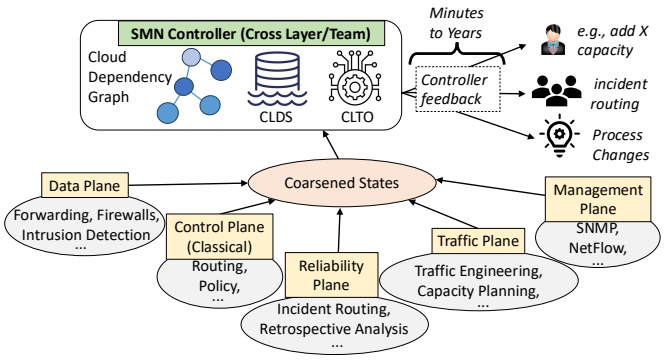

Figure 1: The heart of the Software Managed Network (SMN) vision is a generalized controller that controls all other planes.

The rest of the paper is organized as follows. We introduce our proposal Software Managed Networking (SMN) in §2 as a solution to cross-layer visibility problems, propose a technique called coarsening (§3) to address the scaling challenges in SMN; describe two new examples of coarsening (§4-§5) for reliability and capacity planning; and end by discussing how our proposed SMN architecture can be realized (§6).

2 SOFTWARE MANAGED NETWORKS

"One plane to rule them all." - adapted from J.R.R. Tolkien

We propose Software Managed Networking (SMN) as a novel network management paradigm to centrally maintain and orchestrate network state across the full communication stack — from physical infrastructure to application-level configurations. This ranges from physical cable layouts and submarine cables [34] to customer transport connection data such as goodput [47], and includes failure logs [10] and fine grain traffic information [39]. By integrating network state from the data plane (*e.g.,* packet forwarding), control plane (*e.g.,* traffic engineering), and management plane (*e.g.,* incident response) into a unified, software-driven system, SMNs have the potential to eliminate traditional management silos by enabling holistic visibility and operational control.

Figure 1 sketches the SMN architecture we envision in more detail. The main SMN Controller gets inputs from and

sends feedback to the Data Plane, the classical control plane, and traffic management, reliability and management planes. The SMN Controller has three components, two primary data sources and an optimizer.

Aspects	**SDN**	**SMN**
Scope	Data Plane	All Planes
Timescale	μ seconds to Hours	Minutes to Years
Data Inputs	Structured (Traffic, Topology)	Mixed (Telemetry, Logs)
Outputs	Actions (e.g., add FIB entry)	Actions, Process Changes
APIs	OpenFlow, P4	OpenTelemetry, OpenConfig
Enabling Technologies	NoSQL, Compilers, Optimization	Data Lakes, Generative AI, ML
Managed Layers	L2-L3	L1-L7

Table 1: Comparing SDN to SMN

(1) **Cross-layer and Cross-Team Network State:** SMNs aggregate detailed state information across layers of the networking stack, providing comprehensive visibility into network conditions and behaviors. We call this the Cross-Layer Cross-Team Data Store (CLDS) in Figure 1 where the icon represents a data lake because some data store can be unstructured (e.g., failure logs).

(2) **Dependency Graphs:** A key innovation of SMNs is their explicit maintenance of cross-layer dependency graphs that we call Cloud Dependency Graphs in Figure 1. These graphs capture and detail the complex interdependencies between network states, configurations, and events over the network's entire operational lifetime. These dependency graphs will enable cross-layer analysis and optimizations.

(3) **A Generalized Control Plane.** SMNs significantly expand the foundational principles of Software Defined Networking (SDN). Whereas SDN traditionally focuses solely on centrally programming the forwarding behavior of network switches via forwarding tables, SMNs generalize this approach to encompass centralized management of the Routing Information Base (RIB), Forwarding Information Base (FIB), Management Information Base (MIB) and Diagnostic and Traffic information. Moreover, SMNs distinguish themselves from SDNs by operating over extended temporal horizons. While SDNs concentrate on immediate, short-term operational responses, SMNs adopt a strategic viewpoint, managing network state over months and even years. This long-term perspective enables strategic operations, like capacity planning, reliability engineering, and procedural enhancements.

By maintaining comprehensive network state and detailed dependencies over extended periods, and operating several control loops over different time granularities, SMNs can empower operators to shape the long-term evolution and optimization of complex networks. This aspect is operationalized in Figure 1 by what we call the Cross-Layer, Cross-Team optimizer (CLTO) whose output is a set of feedback either to teams or external agents. For example, for incident response, the time scale can be minutes and the feedback is to the team that is implicated as the cause of the incident; for capacity planning, the time scale can be months or years and the feedback may be to an external provider to provision additional capacity. Table 1 compares the SDN and SMN visions along several relevant dimensions.

How SMNs can mitigate operational challenges. An SMN can address the four war stories in the introduction:

1. Capacity Planning and TE in the Dark: As the SMN is aware through its CLDS of Traffic Engineering decisions at L3 and fiber constraints at L1, it can avoid capacity upgrades of links TE has recently overloaded, and only does so when the overload is sustained over time and fits fiber constraints.

2. Wavelength Modulation and Resilience: As the SMN is aware through its dependency graph of the dependency between optical links and logical inter-DC links, it can trace recurring routing flaps quickly to an aggressive configuration at the optical layer.

3. WAN link flaps impacting cluster traffic: The SMN observes values exceeding threshold in both the cluster and the Wide Area. However, using its Dependency Graph, the SMN computes that most failing cluster probes depend on the wide area. Hence, the SMN routes the incident to the WAN team while informing the cluster team.

4. Database Service Failure: The SMN aggregates alerts by a coarse label (e.g., the service) and find that the alerts of the Database service in aggregate from other services are over threshold, and the Database telemetry also indicates faults, so it sends feedback imputing failure to the database team,

Challenges. The SMN vision, however, has challenges:

Manual Resolution: The war stories above show that many cross-layer problems in clouds are resolved manually in several hours today. SMN should enable various flavors of automation across teams, including using machine learning and generative AI to resolve incidents and optimize resources. This goes beyond mere centralization to enable the right features to be extracted at the right granularity.

Scalability: While attempts like Amazon CloudWatch [3] and OmniTable [33] already provide centralized storage and tracing [11] to correlate data, centralizing this data across

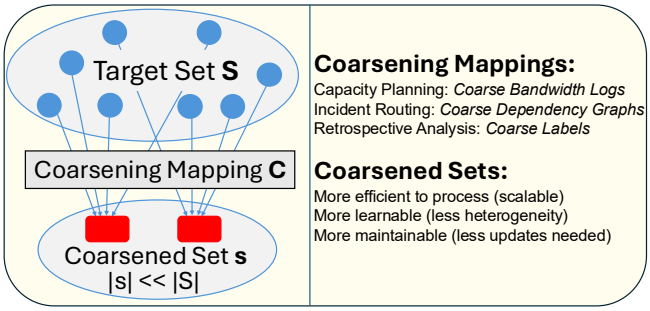

Figure 2: A Coarsening is a mapping between sets that approximately preserves action semantics

teams can take an infeasible amount of storage [36, 43] and bandwidth, but is also expensive to sift through.

Maintainability: In some instances, like dependency graphs (e.g., [28]) across all fine-grained services, the storage for the graph itself is not the bottleneck. What is hard is generating and maintaining the graph because of legacy code and churn.

Organizational Constraints: SMN cannot dismantle the existing successful organizational structure of clouds into teams, but must *augment* them with interfaces to a centralized resource akin to a SDN controller.

Architecture, Interfaces: Like SDN, SMN must go beyond merely centralizing all data. It also requires an architecture and interfaces such as SDN's OpenFlow [12] so that users across teams can query and correlate data.

We propose to tackle the challenge of scalability and maintainability by contracting detailed operational network data into simplified yet informative representations. Effective coarsening must strike a delicate balance between reducing complexity and preserving critical information—ensuring essential insights and dependencies are not lost. We discuss how we plan to achieve this balance in the next section. We then return to the challenge of architecture and interfaces.

3 SMN VIA COARSENING

"All the news that's fit to print." - New York Times motto

A second message of this paper, besides the SMN vision is that we can meet some of the challenges and move towards software managed networks at some loss in fidelity using a technique we call coarsening. Just as the New York Times aims to print all the news that is fit to print, we believe a SMN need only store relevant data, coarsened to fit.

Slightly more formally, given a complex structure S, a coarsening $s = C(S)$ is a succinct mapping of S to a simpler structure s such that $|s| < |S|$ and acting on s is approximately the "same" as acting on S (see Figure 2). For example, answering a set of queries on s could be nearly as accurate as sending the same queries to S. More interestingly, repairing (writing not reading) s, could have approximately the same effect as repairing S, a common example of which is restarting a large system for fast mitigation as in NetPilot [44].

Coarsening is implicit in earlier work. For example, hierarchical routing ([23]) coarsens networks into areas to reduce state at the cost of only approximately optimal routes. Our goal in this paper, however, is to *explicitly* present coarsening as a broad concept that cuts across varied networking tasks such routing, failure detection and capacity planning, while enabling scaling to planetary networks. Using concrete examples, we show why coarsening is useful not just for reducing storage but also for improved maintainability and long term insight. Our goal is to start a conversation about the feasibility of Software Managed Networks, and enable the community to develop other network-specific coarsenings beyond the two described in this paper.

Coarsening is similar but differs from a technique for program analysis called abstract interpretation (AE) [6]. Both work with abstracted descriptions of the state of a system (or program) to gain scalability/feasibility. However, coarsening avoids modeling the system code entirely, and instead works with partial descriptions of system states that arise from system runs. AE also requires an inverse mapping called a concretization function, i.e., sound mapping of the abstract outcomes back to concrete values, to enable downstream analyses crucial in complex program analysis. These differences imply that while AE is sound for failure detection, coarsening may miss (hopefully) rare failures. On the other hand, the added conditions [6] make AE hard to apply to the management of large networks.

While there are many sources of large data in planetary networks such as model training, we describe two illustrative problems where coarsening can be useful: capacity planning and incident response.

Although previous work has aspects of our work (e.g. dependencies [18, 21], coarse fixes [44], bucketing [14]), none introduces the coarsening concept or addresses SMNs.

Our definition of "approximately the same effect" for a coarsening in Figure 2 is vague, and formalizing the general concept remains a challenge. Instead, Table 2 provides an overview of the benefit and the approximation error for the two coarsenings we describe next, together with estimates and simulations that show their promise.

4 COARSE BANDWIDTH LOGS

Our first example of a coarsening is simply to improve *scalability* in storage and computation. Software-defined networks (SDNs) collect rich logs of bandwidth demands on network links, which are critical inputs to both short-term and long-term decision-making problems in the network. In wide-area

Example	Mapping	What's Lost	What's Gained
Coarse BW Logs	Nodes → Meta Nodes	Suboptimal solution	Fast traffic engineering and planning
CDGs	Microservice → team dependency.	Coarser incident routing	Extra Signal for incident routing.

Table 2: Coarsening Examples and Tradeoffs

SDNs, these historical logs are used to forecast future demand [19, 20, 26, 46]. In the short term, traffic engineering controllers use the resulting demand estimates to compute network flow allocations that align with operator goals, ranging from maximizing throughput [45] to enforcing priority fairness [27]. Over longer timescales, bandwidth logs inform capacity planning [39, 49]. For example, operators follow heuristics like augmenting the bandwidth on a link if its utilization consistently exceeds a threshold [38]. Listing 1 shows a representative format of bandwidth logs:

Listing 1: Example uncoarsened bandwidth log: timestamp, source DC, destination DC, bandwidth (Gbps).

```
# Format: ts, src_dc, dst_dc, bw_Gbps
2025-06-01T00:00, us-e1, eu-w1, 1250
2025-06-01T00:05, us-w2, ap-se1, 980
2025-06-01T00:10, us-e1, eu-w1, 1325
2025-06-01T00:15, us-w2, ap-se1, 1010
2025-06-01T00:20, us-e1, eu-w1, 1275
```

As shown in Listing 1, bandwidth logs collected over long time scales in large networks can be massive, with each row capturing the demand between a pair of datacenters in a five-minute time window [19, 20]. As a result, storing and processing them becomes a challenge. We explore the idea of coarsening bandwidth logs along two key dimensions to make them more tractable for SMNs:

Time based coarsening: One approach is to aggregate bandwidth logs over time to reduce input size. For example, traffic engineering controllers can replace per-epoch demand traces, collected over months, with summary statistics (*e.g.,* mean or 95^{th} percentile bandwidth usage) over fixed smaller time windows. More sophisticated variants of this coarsening approach might compute multiple summary statistics over nested time windows to preserve important trends while shrinking the dataset. However, this process risks discarding valuable historical context. For example, a summary over the past month fails to capture the impact of traffic spikes due to seasonal events like federal holidays observed in the previous year [38]. The time windows over which bandwidth logs are coarsened will depend on the nature of traffic patterns — in a time of high churn, we want to coarsen the logs more often to not miss trends.

Topology-based coarsening: A second approach is to coarsen logs over the network graph by grouping nodes into "supernodes". Cloud traffic engineering systems can operate at this granularity by routing traffic between supernodes that represent entire geographical regions (*e.g.,* US east coast), instead of individual datacenters [1, 26]. While this reduces bandwidth log size and makes the flow optimization problem more tractable, both TE and capacity planning applications on the coarsened network graph will find approximate solutions that may be far from optimal. For example, traffic engineering optimization on a coarsened network graph assumes that all traffic from the supernode must be routed along predetermined network edges defined in the coarsened graph [1]. This restriction in the algorithmic search space can lead to suboptimal solutions.

Impact on algorithmic performance. There is a clear tradeoff between coarsening granularity and performance of downstream applications of bandwidth logs. Coarsening hundreds of nodes into, say, ten supernodes will reduce the volume of data logs by an order of magnitude. Moreover, the resulting traffic engineering and capacity planning optimization will be computationally tractable due to small input size and few decision variables. However, very coarse bandwidth logs can make the problem trivially small and reduce the utility of the solutions. For example, coarsening the graph where a supernode represents all datacenters in a continent, will lead to a small topology of 7 nodes. Traffic engineering controllers will then compute optimal allocation of all traffic between any pair of continents (*e.g.,* North America and Europe) which in the coarse graph will use inter-supernode links *i.e.,* subsea cables. The resulting allocations have limited utility, since the routing within the large super nodes is not specified by the optimization.

Key research questions include: 1) Can we find the Pareto frontier between the extent of coarsening (*e.g.,* larger super nodes vs. smaller super nodes) and optimality of algorithms that rely on the coarsened logs? 2) Can we automatically identify which network partitions have more "stable" traffic demand patterns to coarsen only the stable parts?

Potential reduction in log size. Recent work shows that only a small fraction ($\leq 10\%$) of datacenters exchange high volume traffic in cloud wide-area networks [27]. Therefore,

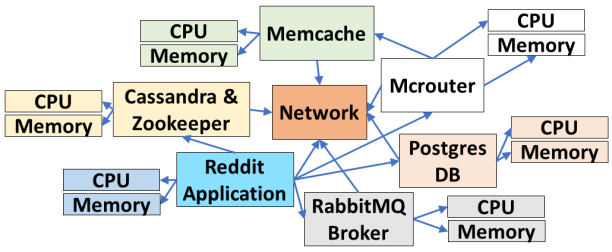

Figure 3: Coarse dependency graph simulating Reddit

in a planet-scale wide-area network of roughly 300 datacenters [26, 46], coarsening the network graph into smaller regions (*e.g.,* US east cost, west coast) will lead to less than 30 high traffic regions, leading to a $10X$ reduction in log size. Combined with time-based coarsening, the reduction factor increases manifold depending on the amount of historical data maintained (*e.g.,* weeks vs. months).

5 COARSE DEPENDENCY GRAPHS

Our second example of a coarsening is to improve *maintainability* for the Cloud Dependency graph of Figure 1. We discuss this coarsening specifically in the context of routing an incident to the right team like the Wide Area versus Cluster war story. A dependency graph contains edges $x \rightarrow y$ if x depends on y at runtime. A coarse-grained dependency graph (CDG) shows dependencies of various services and teams as in Figure 3. By contrast, a fine-grained dependency graph shows dependencies between service components (useful for root causing). While teams *may* maintain their own fine-grained dependency graphs, we propose the SMN only maintain a coarse dependency graph for the cloud (Figure 1).

Tools like Sherlock [28] can extract fine-grained dependencies between services, while tools like CodeScene [41] can infer dependencies between microservices. Fine-grained dependency information at the cloud level is often unavailable and hard to maintain because of different tracing infrastructures and legacy systems. Fortunately, from our experience, engineers can directly sketch the CDG (e.g., cluster probes depend on the WAN) and refine it over time.

Figure 3 shows a CDG for a simulated Reddit implementation. Each node represents a team with edges to other teams it depends on to deliver a service. Coarsening can create false dependencies. For example, in Figure 3 a particular hypervisor failure in the CPU that hosts Cassandra could affect only certain writes to the user profile cache without affecting Reddit's subreddit fetch functionality. While this fine-grain dependency is missed by the CDG, other signals can help route the incident correctly, such as internal health metrics of Reddit (e.g., subreddit cache hits/misses).

We propose a simple technique with which the CLTO in Figure 1 can automate incident routing by leveraging a CDG, and the following derived metric.

Symptom Explainability: Intuitively, symptom explainability of team T is the fraction of symptoms explained using the CDG assuming team T is the only one to cause a failure. More precisely, define the vector of symptoms (i.e., nodes in the CDG who experience symptoms) as an *incident syndrome*. Symptom can be a function (e.g., packet loss > X%) of internal health metrics defined by respective individual teams. We then define *symptom explainability* for team T as the cosine similarity of the incident syndrome to the syndrome if *only* team T caused a failure. This allows for noise, false dependencies and normalizes each team's explainability metric between $[0, 1]$.

Preliminary Results: We simulated 560 fine-grained faults (e.g., hypervisor failure, bad timeouts) from the Revelio Incident Dataset with the open-source Reddit [35] application on the Revelio testbed [10]. We used the symptom explainabilty metric as well as standard internal health metrics [10] from production systems. We implement pairwise reachability probes between application server clusters and application health checks polled by a monitoring agent at 1-minute intervals to mimic metrics used in clouds. We identify 8 "teams" including Network, Application and Infrastructure. For example, an incident caused by a faulty firewall rule should be handled by the network team, and an incident caused by a faulty server should be handled by its microservice infrastructure team.

We use both cosine similarities and internal health metrics as feature vectors input to a Random Forest Classifier to predict the correct team label for a given incident. We re-use the same dataset splits for training, validation, and testing as Revelio does. Our test set only contains incidents that are a result of a root-cause that is never injected in the same way as in the training set. The performance of the Random Forest Classifer for CLTO in routing incidents (amongst 8 teams) on the test set with and without using symptom explainability as a feature improved from 45% to 78% while a purely distributed approach like Scouts [13] was only 22%. This can be attributed to fan-out cause-effect relationships (e.g., a failure in a lower layer causes multiple failures in the x higher layer), which are confounders in distributed approaches that can rely only on internal health metrics of a layer. Centralized approaches like CLTO on the other hand perform much better when this additional context is represented through structures like coarse dependency graphs.

The bottom line is that even imperfect (but easily maintainable) information like a Coarse Dependency Graph is useful in a machine learning context because it provides a strong extra signal in addition to team-internal health metrics. The

same intuition suggests the SMN can route to the Wide Area team in the War story described earlier.

6 REALIZING SMN

Just as realizing SDNs required the development of new networking protocols (*e.g.,* OpenFlow), realizing SMNs will need novel system architectures, data structures, and APIs:

Global data lake: The CLDS in Figure 1 can be a real-time data lake that provides a global view of network components (part numbers, software versions), reliability information (alerts, incidents, telemetry, logs from service teams) and capacity planning telemetry (bandwidth logs, costs). While each team independently manages its own data, and the optimizer (CLTO in Figure 1) consults the CLDS, teams and central leaders can also easily discover and consume data from other teams. To do so requires a (1) A queryable global catalog describing data sets and metadata, including team names, data type (alert/incident/log/telemetry), data schema, units (2) a uniform schema, (3) access control policies, (4) automation that continuously processes real-time telemetry and logs, and (5) policies to retain data. We can leverage recent advances in building large-scale data lakes [2, 7] and instrumentation frameworks such as Open Telemetry [32] to achieve these goals.

Network History store: SMN's data lake, in addition to real-time data, also stores past data to enable historical analysis for reliability and capacity planning by the CLTO in Figure 1. Storing all data for an extended period of time can be expensive. On the other hand, deleting key data may hurt the efforts of learning failure and bandwidth root-cause patterns from the past. The SMN needs sophisticated retention policies: e.g., it can retain all data that are related to incidents for a long period of time. Further, while such positive examples are essential for data-driven automation, they must be balanced by negative examples. The CLDS can also retain a small sample of failure-free data. A more speculative idea is to keep ML models and not logs over very long periods to concisely capture how network patterns evolve with time. These can be viewed as coarsenings in time.

AIOps engine: The CLTO in Figure 1 provides a natural place for developing AIOps[1] solutions. For example, one can: (1) denoise telemetry and logs on injection into the data lake, (2) enrich incidents with metadata such as similar incidents, potential root causes, and fixes learned from retrospective analysis [8], (3) Use generative AI to convert logs into structured inputs for the CLTO (4) learn rules to prioritize and route incidents, (5) take automatic mitigation steps such as rebooting an unhealthy micro-service, or lighting up a fiber. Even though there are existing AIOps efforts [5], SMN's data lake and CLTO makes it easier to experiment with new techniques involving diverse data from multiple layers and teams.

[1]Artificial Intelligence for IT operations

7 CONCLUSION

When Software Defined Networks were first proposed, they were dismissed as unscalable and prone to failures. They have now become mainstream [20, 22]. We believe the time is ripe to revisit centralizing *all* operational data — from assets to incidents to bandwidth patterns — in a centralized repository in a Software Managed Network (SMN) to improve coordination, coherence, pattern learning, and long term planning.

To make this vision a reality, we have proposed the abstract idea of coarsening to reduce storage while preserving operational effectiveness. While cross-layer optimizations and coarsening are both old ideas implicit in much past work (as was SDN), we believe that making coarsening a first class concept, centralizing all data from L1 to L7, and adding a generalized control plane and architectural support can help make SMN a reality. We have provided a few examples of SMN applications and coarsenings. However, just as with SDN, we believe the networking community will find unforeseen cross-layer optimizations and coarsenings for SMN.

Peering dimly into the crystal ball, can mappings from IP links to layer 1 information like submarine cables [34] be used not just for risk modeling [25] but for risk-aware topology design [30] and capacity planning [39] at layer 3. Can Layer 4 information collected in Espresso [47] be used for traffic engineering and capacity planning instead of only to choose BGP peers? Can application level log information at Layer 7 stored in the SMN be used to help an Application Designed Network [50] compiler generate an implementation tailored to the network state? The unstructured information in a SMN (e.g., logs) is a natural fit for generative AI-powered agents for failure response [9] and ticket prioritization [31]. What are the corresponding coarsenings for scalability, learnability, and maintainability? We have no idea, but we hope others in the community will.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers, our shepherd, Brighten Godfrey and Radhika Mysore for their thoughtful feedback. Pradeep Dogga was supported in part by NSF Award No. CNS-1901510 and research grants from Google, Amazon and Cisco. Rachee Singh is supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, NSF Awards No. 2444537 and No. 2435852. George Varghese was supported in part by NSF Award No. 2333587.

REFERENCES

[1] F. Abuzaid, S. Kandula, B. Arzani, I. Menache, M. Zaharia, and P. Bailis. Contracting wide-area network topologies to solve flow problems quickly. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 175–200. USENIX Association, Apr. 2021.

[2] Amazon.com. What is a datalake? https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/, 2022.

[3] AWS. Application and infrastructure monitoring - aws cloudwatch. https://aws.amazon.com/cloudwatch/, 2023.

[4] T. Casciaro, A. C. Edmondson, and S. Jang. Cross-silo leadership. *Harvard Business Review*, May 2019.

[5] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, et al. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1487–1497, 2020.

[6] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '77, page 238–252, New York, NY, USA, 1977. Association for Computing Machinery.

[7] Databricks.com. Introduction to Datalakes. https://databricks.com/discover/data-lakes/introduction, 2022.

[8] P. Dogga, C. Bansal, R. Costleigh, G. Jayagopal, S. Nath, and X. Zhang. AutoARTS: Taxonomy, insights and tools for root cause labelling of incidents in microsoft azure. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 359–372, Boston, MA, July 2023. USENIX Association.

[9] P. Dogga, K. Narasimhan, A. Sivaraman, and R. Netravali. A system-wide debugging assistant powered by natural language processing. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 171–177, New York, NY, USA, 2019. Association for Computing Machinery.

[10] P. Dogga, K. Narasimhan, A. Sivaraman, S. Saini, G. Varghese, and R. Netravali. Revelio: Ml-generated debugging queries for finding root causes in distributed systems. *Proceedings of Machine Learning and Systems*, 4, 2022.

[11] B. Eaton, J. Stewart, J. Tedesco, and N. C. Tas. Distributed latency profiling through critical path tracing: Cpt can provide actionable and precise latency analysis. *Queue*, 20(1):40–79, mar 2022.

[12] O. N. Foundation. Open flow switch specification. https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf/, 2025.

[13] J. Gao, N. Yaseen, R. MacDavid, F. Vieira Frujeri, V. Liu, R. Bianchini, R. Aditya, X. Wang, H. L. , D. Maltz, M. Y. , and B. Arzani. Scouts: Improving the diagnosis process through domain-customized incident routing. In *SIGCOMM*. ACM, August 2020.

[14] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt. Debugging in the (very) large: ten years of implementation and experience. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 103–116, New York, NY, USA, 2009. Association for Computing Machinery.

[15] Google. Google cloud service health. https://status.cloud.google.com/summary, 2023.

[16] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM. ACM, 2016.

[17] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 139–152, New York, NY, USA, 2015. Association for Computing Machinery.

[18] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[19] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 15–26, New York, NY, USA, 2013. Association for Computing Machinery.

[20] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):15–26, aug 2013.

[21] H. Jahanshahi, K. Chhabra, M. Cevik, and A. Baþar. Dabt: A dependency-aware bug triaging method. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, EASE '21, page 221–230, New York, NY, USA, 2021. Association for Computing Machinery.

[22] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, Aug. 2013.

[23] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks; performance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.

[24] R. R. Kompella, A. Greenberg, J. Rexford, A. C. Snoeren, and J. Yates. Cross-layer visibility as a service. In *Proc. of fourth workshop on Hot Topics in Networks (HotNet-IV)*, 2005.

[25] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Ip fault localization via risk modeling. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, page 57–70, USA, 2005. USENIX Association.

[26] U. Krishnaswamy, R. Singh, N. Bjørner, and H. Raj. Decentralized cloud wide-area network traffic engineering with BLASTSHIELD. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 325–338, Renton, WA, Apr. 2022. USENIX Association.

[27] U. Krishnaswamy, R. Singh, P. Mattes, P.-A. C. Bissonnette, N. Bjørner, Z. Nasrin, S. Kothari, P. Reddy, J. Abeln, S. Kandula, H. Raj, L. Irun-Briz, J. Gaudette, and E. Lan. OneWAN is better than two: Unifying a split WAN architecture. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 515–529, Boston, MA, Apr. 2023. USENIX Association.

[28] G. Li, D. Chen, S. Lu, M. Musuvathi, and S. Nath. Sherlock: Unsupervised synchronization-operation inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS 2021, page 314–328, New York, NY, USA, 2021. Association for Computing Machinery.

[29] H. Liu, S. Lu, M. Musuvathi, and S. Nath. What bugs cause production cloud incidents? In *Proceedings of the Workshop on Hot Topics in Operating Systems*, May 2019.

[30] J. C. Mogul, D. Goricanec, M. Pool, A. Shaikh, D. Turk, B. Koley, and X. Zhao. Experiences with modeling network topologies at multiple levels of abstraction. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 403–418, Santa Clara, CA, Feb. 2020. USENIX Association.

[31] Netbrain. Generative ai in network operations. https://www.netbraintech.com/blog/ai-in-network-operations/, 2025.

[32] Opentelemetry.io. Automatic Instrumentation | OpenTelemetry. https://opentelemetry.io/docs/instrumentation/java/automatic/, 2022.

[33] A. Quinn, J. Flinn, M. Cafarella, and B. Kasikci. Debugging the OmniTable way. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 357–373, Carlsbad, CA, July 2022. USENIX Association.

[34] A. Ramanathan and S. Abdu Jyothi. Nautilus: A framework for cross-layer cartography of submarine cables and ip links. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(3), Dec. 2023.

[35] reddit.com. reddit: the front page of the internet. https://reddit.com/, 2022.

[36] K. Rodrigues, Y. Luo, and D. Yuan. CLP: Efficient and scalable search on compressed text logs. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 183–198. USENIX Association, July 2021.

[37] SecurityShelf. Cloud misconfig exposes 3tb of sensitive airport data in amazon s3 bucket: 'lives at stake'. https://securityshelf.com/2022/07/06/cloud-misconfig-exposes-3tb-of-sensitive-airport-data-in-amazon-s3-bucket-lives-at-stake/, 2022.

[38] R. Singh, S. Agarwal, M. Calder, and P. Bahl. Cost-effective cloud edge traffic engineering with cascara. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 201–216. USENIX Association, Apr. 2021.

[39] R. Singh, N. Bjorner, S. Shoham, Y. Yin, J. Arnold, and J. Gaudette. Cost-effective capacity provisioning in wide area networks with shoofly. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 534–546, New York, NY, USA, 2021. Association for Computing Machinery.

[40] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill. Radwan: Rate adaptive wide area network. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 547–560, New York, NY, USA, 2018. Association for Computing Machinery.

[41] A. Tornhill. Microservice dependencies - visualization. https://codescene.com/blog/visualize-microservice-dependencies-in-team-context/, 2022.

[42] A. Viswanathan, E. C. Rosen, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, Jan. 2001.

[43] J. Wei, G. Zhang, Y. Wang, Z. Liu, Z. Zhu, J. Chen, T. Sun, and Q. Zhou. On the feasibility of parser-based log compression in Large-Scale cloud systems. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 249–262. USENIX Association, Feb. 2021.

[44] X. Wu, D. Turner, C.-C. Chen, D. A. Maltz, X. Yang, L. Yuan, and M. Zhang. Netpilot: automating datacenter network failure mitigation. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, page 419–430, New York, NY, USA, 2012. Association for Computing Machinery.

[45] Z. Xu, F. Y. Yan, R. Singh, J. T. Chiu, A. M. Rush, and M. Yu. Teal: Learning-accelerated optimization of wan traffic engineering. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 378–393, New York, NY, USA, 2023. Association for Computing Machinery.

[46] C. yao Hong, S. Mandal, M. A. Alfares, M. Zhu, R. Alimi, K. N. Bollineni, C. Bhagat, S. Jain, J. Kaimal, J. Liang, K. Mendelev, S. Padgett, F. T. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan. In *SIGCOMM'18*, 2018.

[47] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 432–445, New York, NY, USA, 2017. Association for Computing Machinery.

[48] Z. Zhong, M. Ghobadi, A. Khaddaj, J. Leach, Y. Xia, and Y. Zhang. Arrow: Restoration-aware traffic engineering. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 560–579, New York, NY, USA, 2021. Association for Computing Machinery.

[49] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 258–271, New York, NY, USA, 2021. Association for Computing Machinery.

[50] X. Zhu, W. Deng, B. Liu, J. Chen, Y. Wu, T. Anderson, A. Krishnamurthy, R. Mahajan, and D. Zhuo. Application defined networks. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, page 87–94, New York, NY, USA, 2023. Association for Computing Machinery.