Toward Scaling Network Emulation using Topology Partitioning

Ken Yocum, Ethan Eade, Julius Degesys
David Becker, Jeff Chase and Amin Vahdat *
Department of Computer Science
Duke University
{grant,eade,degesys,becker,chase,vahdat}@cs.duke.edu

Abstract

Scalability is the primary challenge to studying large complex network systems with network emulation. This paper studies topology partitioning*, assigning disjoint pieces of the network topology across processors, as a technique to increase emulation capacity with increasing hardware resources. We develop methods to create partitions based on expected communication across the topology. Our evaluation methodology quantifies the communication overhead or* efficiency *of the resulting partitions. We implement and contrast three partitioning strategies in ModelNet, a large-scale network emulator, using different topologies and uniform communication patterns. Results show that standard graph partitioning algorithms can double the efficiency of the emulation for Internet-like topologies relative to random partitioning.*

1 Introduction

Network emulation is a powerful tool for testing and evaluating complex distributed network services and systems. A network emulator subjects the communication of applications to the characteristics of network topologies. Emulation offers the promise of accurate, reproducible, and realistic evaluation scenarios. Recent work in network emulation [12, 13] has shown that it is an attractive alternative to simulation for running controlled large-scale experiments.

The principle job of a network emulator is to delay incoming network traffic according to the characteristics of the network or *target* topology. Current network emulators can handle a gigabit of traffic for simple topologies using a single server [12]. However Internet-like topologies with large diameters, competing cross-traffic, small packets, or large-scale peer-to-peer systems with thousands of instances exceed the emulation capacity of a single node.

This work integrates topology partitioning into ModelNet [12], a large-scale network emulator. We study automatically partitioning the target topology across a server cluster to increase *emulation capacity*, the number of packets correctly delayed per unit time. Partitioning assigns disjoint sets of network links from the target topology across the machines responsible for emulation. However, like many distributed computing problems, there is a communication cost to harnessing multiple machines. Packets that are routed between partitions require *cross-core* communication to forward state from one machine to another. This is the principal source of overhead for distributed emulation. A good partitioning minimizes cross-core communication while balancing load across the emulation nodes. Though the problem is NP-hard [3], heuristics yield good partitions in practice [4].

This work capitalizes on the fact that network emulation is a simple distributed computation of packet delays, and can leverage traditional parallel computing scaling techniques. Partitioned network simulation also causes cross-core communication, but incurs additional costs to synchronize time between partitions. Though topology partitioning is recognized as a method for scaling network simulation [6], most simulators perform partitioning by hand [9, 16, 11] or avoid partitioning overheads by running on tightly-coupled multiprocessors [10]. Recent work in automatically mapping experiments across wide-area network testbeds uses simulated annealing to solve a related constraint satisfaction problem [8]. Because network emulation runs in real time, the cost of partitioning is a direct function of the cross-core communication incurred when responsibility for emulating a given packet is passed from one core to another. Our hope is that these techniques may eventually apply to partitioning large network simulations as well.

This paper presents a general approach to partitioning network topologies automatically, using standard graph partitioning algorithms. We study two algorithms: one creates partitions based purely on graph structure while the other creates partitions based on observed inter-partition packet flows. We develop techniques to run the second "intelligent" partitioner based on *expected* communication patterns. We compare these algorithms to random partitions

*This work is supported in part by the U.S. National Science Foundation (EIA-9972879, EIA-0088078, and CCR-03-06490) and IBM.

with respect to their ability to minimize cross-core communication and balance load. Our results evaluate the partitioning schemes across artificial and Internet-like iNet [14] topologies for uniform traffic patterns.

2 Distributed Network Emulation

A network emulation is defined by the application generating network traffic and a *target* network topology. Physical hardware resources, a server cluster or a multiprocessor, emulate the characteristics of the target network topology. The physical machines are split into two sets: *edge nodes* run the application workload and *core nodes* emulate the target network topology. Core nodes emulate each hop of a packet's path from source to destination in the target topology. The target topology is a weighted and undirected graph $G_t(V, E)$. Each edge $e \in E$ represents a link in the target topology, and has a specified bandwidth, latency, loss probability, and queue management policy (drop tail, RED, etc.,). In this paper we refer to edges as *pipes*, as they also represent packet queues in the network emulator. The pipe is the unit of work for network emulation.

Topology partitioning assigns disjoint pieces of the target topology among k core nodes. Figure 1 shows an example of partitioning a target topology between two core nodes. Emulation cost has three components, each based on the flow of packets into the emulator from the application: λ_{edge}, λ_{hop}, and λ_{xc}. λ_{edge} is the application communication rate from the edge nodes. λ_{hop} is the number of packet hops that core nodes emulate per second. λ_{xc}, *cross-core* traffic, is the traffic caused by partitioning the topology. Cross-core hops decrease emulation capacity, decrease accuracy by adding real time delay to the emulated path, and increase physical network link stress as packets move between cores.

We use a simple metric, *effi ciency* (σ), to measure the quality of the topology partitioning. The total packets per second core X processes is λ_{X_t}. The portion of λ_{X_t} that comes from edge nodes is $\lambda_{X_{edge}}$. $\lambda_{X_{xc}}$, the cross-core flow to core X, is $\lambda_{X_t} - \lambda_{X_{edge}}$. If k is the number of cores in the system, then $\sigma = \frac{\sum_{i=1}^{k} \lambda_{i_{edge}}}{\sum_{i=1}^{k} \lambda_{i_t}}$. This represents the percentage of packets dedicated to edge traffic. The goal of network topology partitioning is to drive this measure toward one.

3 Topology Partitioning

Graph partitioning is a difficult, long-standing computational problem. It has applications to VLSI design, sparse matrix-vector multiplication, and parallelizing scientific algorithms. The general k-way partitioning problem is described by a graph $G(V, E, W_V, W_E)$. Where W_V and W_E represent vertex and edge weights respectively. The output of partitioning G consists of subsets of *vertices*, $V_1, V_2, ..., V_k$, where $V_i \cap V_j = \emptyset$ for $j \neq i$, and $\bigcup_i V_i = V$. The goal is to balance the sum of vertex weights for each V_i, and minimize the sum of edge weights whose incident vertices belong to different partitions.

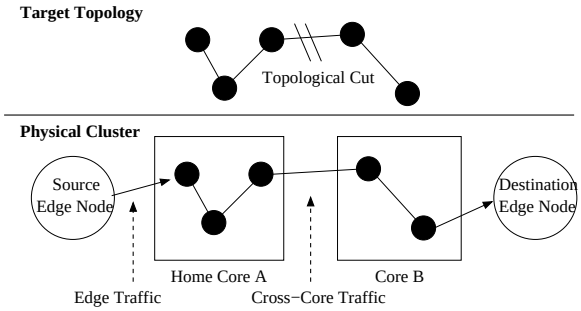

Figure 1. In a network emulator edge nodes (circles) host the distributed application and send their data to core nodes (squares). The cores cooperate to subject the traffic to the bandwidth, latency, and loss of the target network topology. This figure illustrates one possible partitioning of the target topology onto a network emulator with two core nodes.

Note that this formalization of the problem returns a *vertex* partition, but, for network emulation, we want an *edge partition*, an assignment of pipes to core nodes. Further, the resulting partition is based directly on edge weights that represent observed packet flows through the network. Our contributions here are twofold: a simple transform on G_t to create a vertex partitionable graph, and an algorithm to weight the graph based on an expected communication pattern derived from the bandwidth characteristics of the target topology.

We use two graph partitioning schemes: k-cluster [1] and METIS [4]. The advantage to k-cluster is that it requires no transform to G_t, and returns an *edge* partition. k-cluster partitions G_t into k connected components of equal size by iteratively building connected components through breadth-first searches. The shortcoming of this approach is that edges are treated equally, when in fact communication patterns favor some edges and routes more than others.

In contrast, sophisticated partitioners that minimize cross-core packet flows can adapt to specific application communication characteristics. We use the multi-level graph bisection algorithms in METIS [4], a graph partitioning library. This package uses heavy-edge matching to coarsen the graph, a greedy graph growing partitioner (a partitioning algorithm that grows connected components

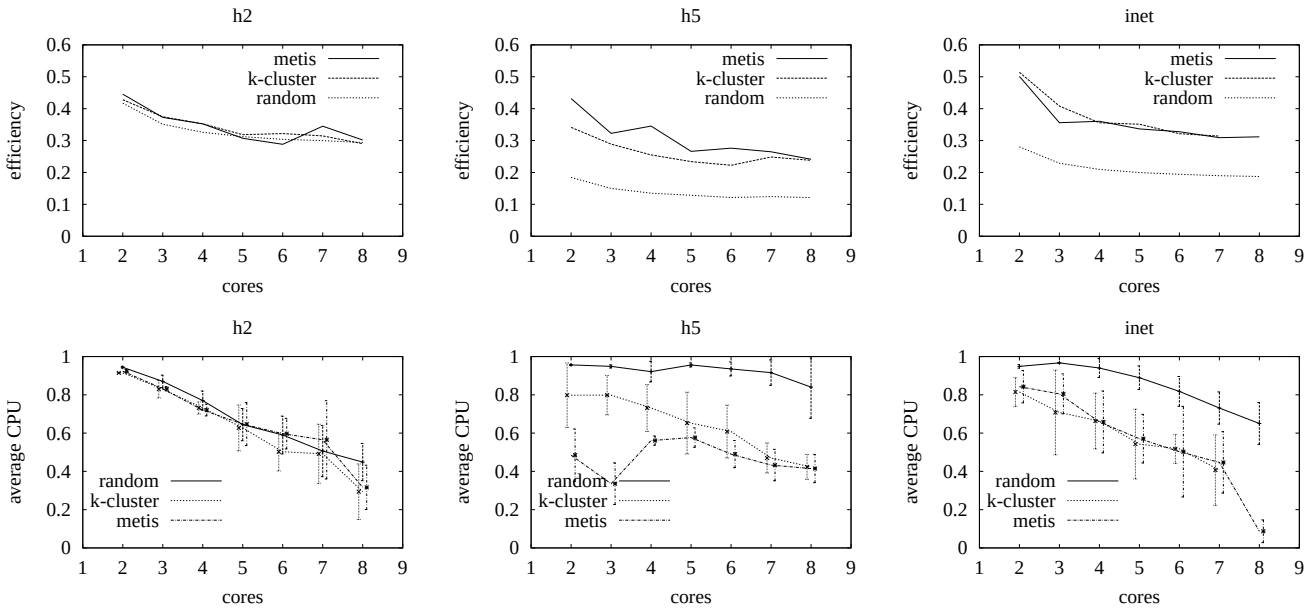

Figure 2. These graphs show emulation efficiency and average CPU utilization (with horizontally staggered standard deviations) for $h = 2$, $h = 5$ and an iNet topology for random, k-cluster, and METIS partitioning. For larger topologies ($h = 5$ and iNet), a smart partitioning scheme yields significant benefits. For iNet, METIS provides a 65% efficiency improvement over random for an 8-way partition.

like k-cluster but considers flow contributions of added edges), and uses a modified Kernighan-Lin (KL) [5, 2] to refine the graph. We considered a sophisticated approach called spectral partitioning [7], however our initial experiments indicated that partitions were no better than METIS and often took longer to compute. We have verified that METIS calculates the optimal solution for small graphs we solved through brute force (up to 90 vertices).

These partitioners require a vertex-partitionable graph. A simple transform on $G_t(V_t, E_t)$ creates a *flow graph* $G_f(V_f, E_f, W_{Vf}, W_{Ef})$ whose vertex partitions map surjectively onto edge partitions of G_t. First, create a vertex v_i in flow graph G_f for each edge e_i in G_t. If two edges e_1 and e_2 in G_t are incident on the same vertex, then create an edge in G_f between the two vertices v_1 and v_2 in G_f. This represents a potential flow crossing both e_1 and e_2; the edge weights W_{Ef} represent the flow between two edges in G_t. The vertex weights, W_{Vf}, are the flow through an edge in G_t and represent the cost of emulating pipe v. G_f has $|E_t|$ vertices and $O(|E_t| * 2 * d)$ edges (d is the maximum vertex degree in G_t).

Next, we weight the edges and vertices of G_f based on an *expected*, uniform communication pattern derived from the link capacities in the original target topology, G_t. The algorithm weights the graph assuming an all-pairs communication pattern and that each pair creates a maximizing TCP-like flow that attains throughput equal to the bottleneck link between source and destination. The resulting flow graph weights are conservative; they approximate worst-case network conditions.

4 Results

We added support for topology partitioning to the ModelNet network emulator [12]. ModelNet edge nodes run unmodified applications and operating systems; they route their network traffic to a set of core nodes that emulate a target topology. Each edge node may host multiple instances of an application. ModelNet core nodes employ payload caching, a technique (based on [15]) that reduces physical link stress by sending packet descriptors between cores. Packet payloads are cached at the core emulating the path's first hop. This reduces the bytes of cross-core traffic by an order of magnitude when using a 1500 byte MTU.

We compare three partitioning strategies, random, k-cluster, and METIS, on an Internet-like topology generated by iNet [14], a 4000 node graph with 200 stubs and 400 clients. We also use an artificial topology that allows us to increase complexity in a controlled manner to evaluate the impact on the partitioning scheme. It is an n-ary tree where

the roots of 4 trees hang off a central mesh. This topology guarantees the path length between clients in separate trees will be $2 * h + 1$, where h is the height of the tree. Increasing h increases the diameter and the number of links shared by multiple routes. We use $h = 2$ (h2), and $h = 5$ (h5) topologies. The workload consists of 400 maximizing TCP streams, generated with user-level microbenchmark applications, whose initial hops are constrained to 3Mb/s. We run k-cluster five times and use the best partition.

The top row in Figure 2 shows the efficiency of each partitioning scheme across the three topologies as a function of core nodes (partitions). The trend for all topologies is a decrease in efficiency. Additional partitions make it increasingly difficult to find good partitions, and suggests a point of diminishing returns using topology partitioning. The small diameter and the high degree of routes sharing edges makes the simple h2 topology amenable to even random partitioning. In contrast, random partitioning fairs poorly for topologies with larger diameters or Internet-like structure. In fact, at least one core was saturated for every random partition on the h5 topology. METIS more than doubles efficiency on the h5 topology with 4 cores relative to a random partitioning (36% improvement compared to k-cluster). The bottom row in Figure 2 shows that this halves the CPU utilization, potentially doubling emulation capacity.

Though both k-cluster and METIS show higher efficiencies for iNet versus h5, the CPU loads are more unbalanced. k-cluster ignores load on the cores, but METIS may sacrifice load balancing for increased efficiency. This is important for emulators such as ModelNet, where cores running beyond saturation may produce inaccurate results [12]. We were surprised at the performance of k-cluster relative to METIS. However the uniform traffic pattern of these experiments does not stress the weakness of k-cluster; it does not use flows when determining a partition. In contrast, METIS-style partitioning can take advantage of flow graphs weighted with asymmetric or dynamic communication patterns, and increase emulation capacity.

5 Conclusion

This paper studies the use of topology partitioning to scale network emulation. The techniques were implemented and evaluated in ModelNet, a large-scale network emulator. We developed methods to apply standard graph partitioning algorithms to network topologies. A simple algorithm transforms the network topology into a vertex-partitionable graph, and this graph is weighted using an assumption of static, uniform TCP-like traffic. Our results indicate standard partitioning algorithms can double emulation capacity versus a random assignment for Internet-like network topologies. However static flow graph weightings do not take into account asymmetric or dynamic communication patterns. We are currently designing dynamic partitioning schemes to meet the challenges of emerging large-scale adaptive applications (peer-to-peer, overlays, CDNs, etc.).

References

[1] P. Ciarlet, Jr and F. Lamour. On the validity of a front oriented approach to partitioning large sparse graphs with a connectivity constraint. Technical Report 94-37, Computer Science Department, UCLA, Los Angeles, CA, 1994.

[2] D. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[3] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1989.

[4] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientifi c Computing*, 20(1):359–392, 1998.

[5] B. W. Kernighan and S. Lin. An effi cient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):291–308, 1970.

[6] D. M. Nicol. Scalability,locality,partitioning, and synchronization in PDES. In *Proceedings of the IEEE Workshop on Parallel and Distributed Simulation*, pages 4–11, June 1998.

[7] A. Pothen, H. D. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11:430–452, 1990.

[8] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *In Computer Communications Review*, 2003.

[9] G. F. Riley, R. Fujimoto, and M. H. Ammar. A generic framework for parallelization of network simulations. In *Proceedings of the 1999 International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 1999.

[10] R. Simmonds, R. Bradford, and B. Unger. Applying parallel discrete event simulation to network emulation. In *The 14th Workshop on Parallel and Distributed Simulation (PADS 2000)*, May 2000.

[11] B. K. Szymanski, A. Saifee, A. Sastry, Y. Liu, and K. Madnani. Genesis: a system for large-scale parallel network simulation. In *Proceedings of 16th Workshop on Parallel and Distributed Simulation (PADS)*, Washington, DC, May 2002.

[12] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI)*, December 2002.

[13] B. White, J. Lepreau, L. Stoller, R. Ricci, S. G. M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI)*, December 2002.

[14] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, Computer Science Department, University of Michigan, Ann Arbor, MI, 2002.

[15] K. Yocum and J. Chase. Payload caching: High-speed data forwarding for network intermediaries. In *Proceedings of USENIX Technical Conference*, 2001.

[16] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.