Dynamis: Dynamic Overlay Service Composition for Distributed Stream Processing

Farshad A. Samimi and Philip K. McKinley
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48823, USA
farshad@ieee.org, mckinley@cse.msu.edu

***Abstract*— This paper addresses the problem of mapping software services onto an overlay network, specifically, the probing to locate suitable nodes on which to instantiate or configure data processing operators. We propose a distributed algorithm, called Dynamis, that can improve existing probing algorithms. Experimental results on the PlanetLab testbed show that Dynamis can dramatically reduce probing overhead while producing high-quality services.**

I. Introduction

Processing data streams as they are delivered across a network is essential to many applications. In the case of data gleaned from sensor networks, for example, processing the data as it is collected supports real-time applications and facilitates later searching and analysis of data repositories. While the components of some data processing services are relatively static, others depend on dynamics in the user population and their queries, in which case the services need to be dynamically composed and reconfigured as conditions change. Realizing this functionality has been facilitated by the advent of *overlay networks*, in which end hosts form a virtual network atop a physical network. The presence of *hosts* along the paths between endpoints enables intermediate processing of data streams, without modifying the underlying network protocols or router software.

In this paper, we focus on the problem of mapping distributed services onto an overlay network. This functionality is an integral part of any overlay-based streaming framework and typically requires a probing mechanism to locate suitable nodes on which to instantiate new data processing operators [1], or to reconfigure and possibly share existing operators [2]. The probing protocol should incur minimal traffic overhead while producing a high-quality mapping of services onto the overlay infrastructure. The quality can be measured in terms of metrics such as end-to-end delay, load balance, security, and cost.

The contributions of this study are threefold. First, we propose *distributed selection*, an optimization technique that supports the design of efficient probing mechanisms. We demonstrate that applying distributed selection to probing algorithms can significantly reduce probing overhead. Second, we introduce an extensible algorithm based on distributed selection, called Dynamis, to realize efficient probing for overlay service composition. Third, we report results of an experimental study on the PlanetLab Internet testbed, where we assess the performance of Dynamis and other service composition algorithms.

The remainder of this paper is organized as follows. Section II provides background and discusses related work. Section III formulates the problem of service composition and probing. Section IV introduces Dynamis, and Section V describes the experimental investigation. Finally, in Section VI we conclude the paper and discuss future directions.

This work was supported in part by the U.S. Dept. of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, and in part by National Science Foundation grants EIA-0000433, EIA-0130724, and ITR-0313142.

Farshad Samimi is currently with GoldSpot Media, Inc. This research was conducted while he was at Michigan State University.

II. Background and Related Work

The service composition problem arises in distributed environments where the system needs to set up and bind a number of entities in order to realize services [3]. With the emergence of overlay networks and adaptive middleware technologies [4], dynamic composition of overlay services has recently attracted considerable attention [5]–[7]. Overlay networks provide a chassis on which to deploy services, and adaptive middleware enables dynamic instantiation and configuration of distributed service components. Overlay service composition is particularly useful in distributed processing of data streams [8], [9].

A fundamental issue in overlay service composition is the probing method to locate and select a set of nodes on which to execute stream processing operators (service elements). Researchers have investigated two main aspects of this problem: locating nodes on which to execute the operators [1], [10]–[12] and sharing of services and processed data [2], [13]. In both cases, most prior research has addressed situations where services already exist in the network and need to be connected together to form a suitable *service graph*. Gu et al. [10] introduced SpiderNet, a peer-to-peer service composition framework that performs distributed bounded probing. A key property of the SpiderNet algorithm is that probes are distributed only to nodes capable of executing the required functions. Pietzuch et al. [1] proposed SBON, a protocol that locates suitable nodes on which to place stream operators. The SBON design is based on a "cost space," a multi-dimensional metric where the distance between nodes is an estimate of the cost of routing between them (in terms of desired measurements such as latency and processing power). Liang and Nahrstedt [11] have addressed the problem where data streams from multiple sources are processed and aggregated

to be delivered to multiple destinations. Finally, Repantis et al. [2] proposed the Synergy framework as a means to reuse existing streams and processing components when composing services. These studies and others have significantly advanced the areas of service composition and data stream processing.

Dynamis complements the above approaches by realizing a generic optimization technique based on distributed selection. As we will show, distributed selection can be applied to existing probing protocols, such as those in SpiderNet [10] and SBON [1], in order to reduce probing costs. We implemented and evaluated Dynamis using Service Clouds [14], an overlay-based infrastructure to compose autonomic communication services. A service cloud can be viewed as a collection of hosts whose resources are available to enhance services (e.g., in terms of fault tolerance and quality of service) transparently to the endpoints. Effectively, overlay nodes provide a "blank computational canvas" on which services can be instantiated and reconfigured as needed. Here, we use different probing algorithms in Service Clouds and evalutate the differences in the resulting service mappings.

III. Problem Formulation

In this section, we provide basic definitions and formally state the probing problem for distributed service composition.

Service Element (Operator). A service element $S = \{F, R, I, O\}$ is a service entity executing on a single node, where F specifies the set of functions carried out by the service; R defines the resource requirements of the service, for example, {memory, processing power, output bandwidth}; I specifies acceptable input, for example, {bit rate, resolution} in a video stream; and O states the generated output specifications, for example, {size, fps} in a video stream.

Service Path. A service path P is an alternating directed sequence $P : n_0, l_0, n_1, l_1, ..., n_n, l_n (n > 0)$ of overlay nodes and overlay links l_i, where $l_i = (n_{i-1}, n_i)$, such that each node executes one or more service elements (S_i) each time it is visited on the sequence. [1] Figure 1 depicts an example service path S consisting of three service elements distributed between two endpoints. We note that it is necessary to specify *both* nodes and links in the path, since an overlay network may be multichanneled, with multiple overlay links following different physical paths between the same two nodes [15].

Service Graph. A service graph $\lambda = \cup\{P_i\}$ comprises the union of one or more connected service paths. Figure 1 shows a service graph that forms a multicast tree.

Service Graph Quality. The quality of a service graph is the end-to-end quality observed at the endpoints. The quality measurement is domain specific and may include overlay stretch properties such as end-to-end delay and packet loss, or non-functional aspects such as security, reliability, and cost.

Hosting. We say that a node can host a service element if that node has available resources to execute a service element and satisfy domain-specific criteria of the service graph, such as reliability, security, and end-to-end delay.

[1] In graph theory, such a traversal is called a *walk*, and a *path* is a walk in which no vertex is repeated. Since in the service composition literature the term *service path* is common and used loosely, we also use this term.

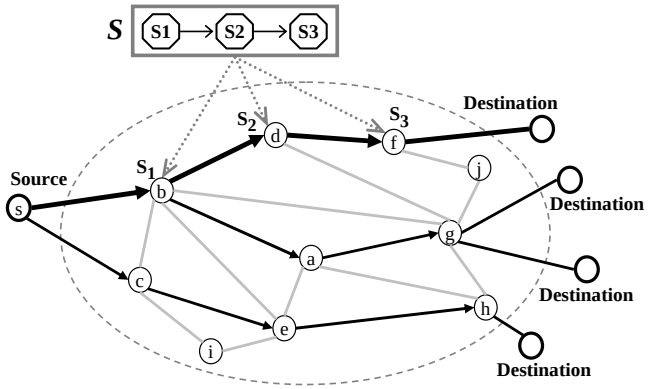


Fig. 1. Service graph example (highlighting a service path).

Comparable Service Graphs. Two service graphs are comparable if they map the same functionality onto the overlay network. In Figure 2, the service paths at the bottom of the figure, $a - g$ and $c - e - h$, both host service elements S_1, S_2, and S_3, so they are comparable; they are not comparable to the $d - f$ graph, which hosts only services S_2 and S_3.

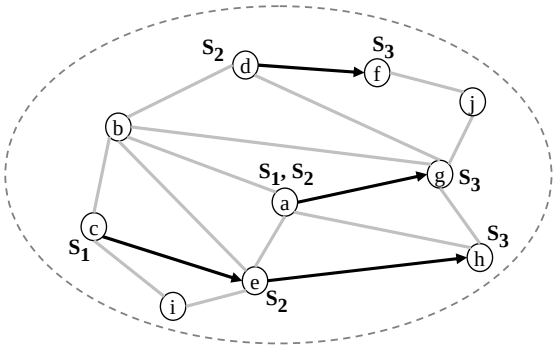


Fig. 2. Examples of service graphs.

This work focuses on composing service *paths*; the general problem of composing service *graphs* is part of our ongoing work. Figure 3 shows examples of two overlay service paths. In these figures, the notation $\{S_i, ..., S_j\}$ represents an unordered set of service elements, that is, functionality of members is commutative. The notation $(S_i, ..., S_j)$ represents an ordered set of service elements, that is, the functionality of each member depends on the previous one and is non-commutative.

Formally, we can state the problem addressed in this paper as follows: Given an overlay graph G of n nodes and a service specification S, find a path P in G such that P can host S.

IV. Dynamis Probing Algorithm

In a probe-based approach to service composition, multiple probes are sent to find service path candidates. A probe contains the requirements of the service path, including the ordering constraints, resource requirements, and expected end-to-end quality. The Dynamis algorithm is based on *distributed selection*, which applies the principle of optimality, namely, that in an optimal sequence of decisions or choices, each subsequence must also be optimal [16]. We observe that service path composition satisfies the principle of optimality.

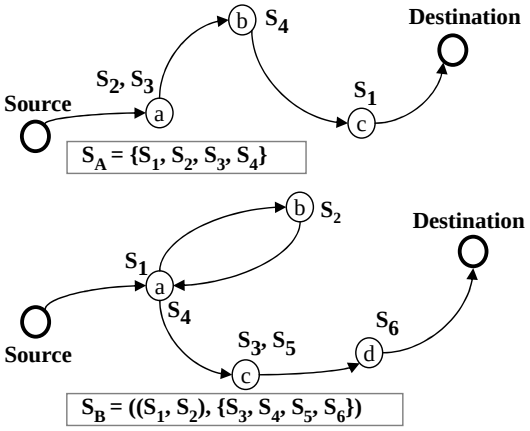


Fig. 3. Service ordering constraints and corresponding service paths.

That is, for any overlay node on an optimal service path, the two partial service paths from that node to the endpoints must also be optimal. We use this observation to design a probing algorithm in which an overlay node drops or forwards a probe based on the quality of the partial service paths found earlier.

Figure 4 depicts the basic operation of the Dynamis probing mechanism, which generalizes an algorithm proposed by Tang and McKinley [15] to construct multipath connections in overlay networks. One of the endpoints initiates probing (path-explore process) by sending the probe for a service path to a subset of its neighbors in the overlay network, according to a predefined branching factor. Thereafter, probes attempt to find their way to the other endpoint. In the algorithm presented here, the destination endpoint (arbitrarily and without loss of generality) starts the path-explore process.

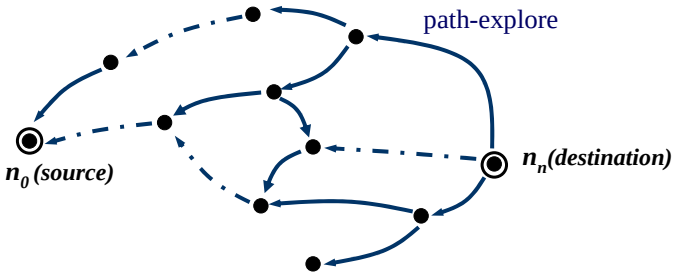


Fig. 4. Basic operation of the composition probing algorithm.

To measure quality of a service path λ, denoted $\psi(\lambda)$, we extend the load balancing metric from Synergy [2] to include the round-trip time as a factor. Specifically,

$$\psi(\lambda) = \omega_p \sum \frac{p_{s_i}}{q_{v_i} + p_{s_i}} + \omega_b \sum \frac{b_{s_i}}{c_{v_i} + b_{s_i}} + \omega_d \frac{D}{D_{max}} n \quad (1)$$

where the terms of the formula are defined in Table I. This metric quantifies the quality of a service path based on the processing and bandwidth load of overlay nodes on the service path, as well as its end-to-end overlay delay. The smaller the $\psi(\lambda)$ value, the better the quality of the service path.

The key property of the Dynamis algorithm is that rather than performing selection only at an endpoint, the selection is distributed. An overlay node forwards a probe only if it describes a partial service path of significantly better quality than the quality of *comparable* service paths described by probes forwarded previously. As we shall demonstrate in Section V, this approach can dramatically reduce the overhead of probing while retaining high quality.

TABLE I

Notation	Meaning
λ	service path
p_{s_i}	processing resource required for service element S_i
q_{v_i}	residual processing capacity on node v_i
b_{s_i}	uplink bandwidth required for service element S_i
c_{v_i}	residual uplink bandwidth on node v_i
D	end-to-end delay of an overlay path
D_{max}	maximum acceptable end-to-end delay
n	number of nodes in the service path
$\omega_p, \omega_b, \omega_d$	experimental cofactors

Algorithm Sketch. Upon receiving a path-explore probe, each node n_i inspects the probe and performs one of the following actions (details can be found in [17]):

(i) Node n_i drops the probe, in any of the following conditions:

(a) The service path violates the expected quality (e.g., end-to-end delay) at this point.

(b) Node n_i cannot satisfy resource requirements for hosting at least one of the service elements that can be added to the partial service path explored to this point (hosting).

(c) The quality of the partial service path explored so far is not *significantly* better (e.g., by at least 5%—as specified in the configuration) than the quality of the best *comparable* partial service path described by a probe already forwarded by node n_i.

(d) The quality of the partial service path explored so far is not better than the quality of a comparable partial service path described by a probe currently buffered to be forwarded.

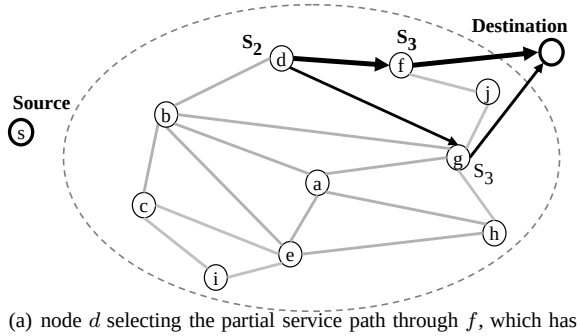
(ii) Otherwise, node n_i:

(a) Updates the probe, adding itself to the partial service path described by the probe.

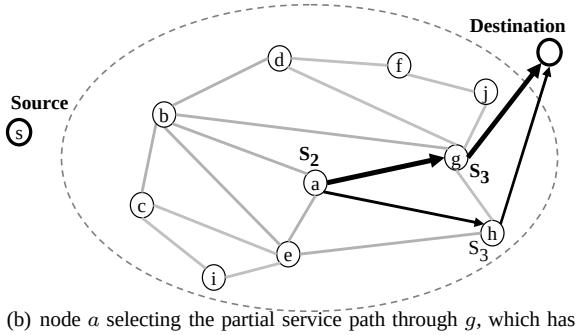
(b) If the partial service path achieves the requested service path, node n_i announces a candidate service path. If node n_i is the target endpoint of the probe, then this announcement is local; otherwise, node n_i forwards the probe to the target endpoint.

(c) Otherwise, node n_i buffers the probe, replacing any probe in the buffer that describes a comparable partial service path. Node n_i periodically forwards all probes in the buffer; this period is called an *epoch*. Node n_i forwards a probe to a subset of "qualified" neighbors. Criteria for qualification include trust relationship, cost, and availability of a particular functionality.

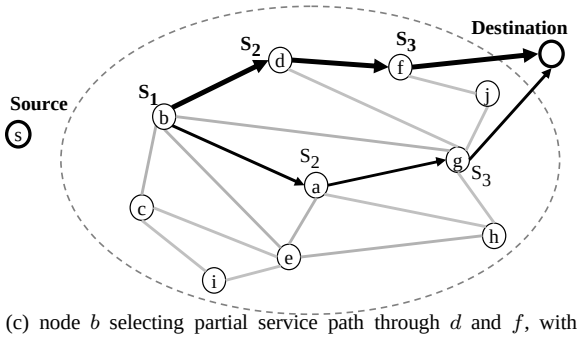
Example. Figure 5 demonstrates a simplified run-time operation of the Dynamis algorithm, in which the service path $S = (S_1, S_2, S_3)$ is being mapped to overlay nodes. In Figure 5(a) two comparable partial service paths have been found up to node d during a distributed selection epoch. The algorithm forwards only the probe describing the path of highest quality, and then only if the quality is significantly better than the best comparable service path forwarded in



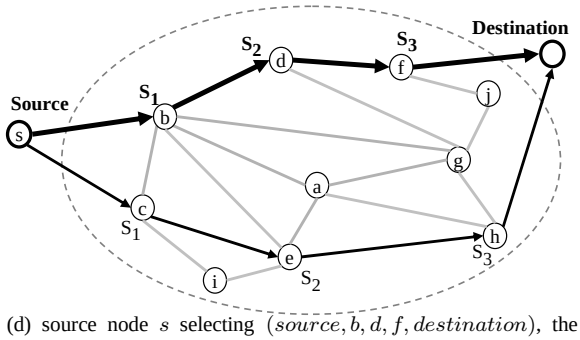
(a) node d selecting the partial service path through f, which has higher quality than the comparable one through g



(b) node a selecting the partial service path through g, which has higher quality than the comparable one through h



(c) node b selecting partial service path through d and f, with higher quality than the path through a and g



(d) source node s selecting $(source, b, d, f, destination)$, the candidate service path with highest quality

Fig. 5. Example run of the Dynamis probing algorithm: mapping $S = (S_1, S_2, S_3)$ to the overlay network.

a previous epoch. Let us assume that S_2 can be hosted by node d, so node d forwards $(d, f, destination)$. In a similar way, in Figure 5(b), $(a, g, destination)$ is selected. Then, in Figure 5(c) node b receives two comparable partial service paths from nodes d and a. Again, applying distributed selection (and assuming S_1 can be hosted by node b), only the partial path with higher quality is selected if both of the probes are received in the same epoch. If the probes are received during different epochs, then the one received later is selected only if it has significantly better quality. Now, let us assume that only $(b, d, f, destination)$ is selected, which is forwarded to the source node as a candidate service path. Eventually, the source node selects the candidate with the highest quality, $(source, b, d, f, destination)$ in this example, and maps the service elements onto the corresponding overlay nodes. In approaches that do not use distributed selection, nodes typically forward all of the probes which they receive.

Parameters. Table II gives the parameters used in the algorithm. T is the buffer time-out period, or *epoch* duration; when the buffer is empty and a probe is placed in it, a timer starts that flushes the buffer after T milliseconds. $\overline{B}$, $\overline{H}$, and $\overline{W}$ are branching factor parameters that control the overhead of the probing in three respective ways: budgeted, limited-hop, and bounded. In *budgeted* forwarding, each node forwards a probe to $\overline{B}$ qualified neighbors selected at random. In limited-hop forwarding, a probe is discarded if it has not found a service path after traversing $\overline{H}$ overlay nodes. In *bounded* forwarding, each node forwards at most $\overline{W}$ probes for a service composition session. Finally, $\underline{Q}$ specifies the minimum service path quality improvement expected, relative to the quality of a comparable service path in a probe forwarded previously, in order to forward a probe in question.

TABLE II

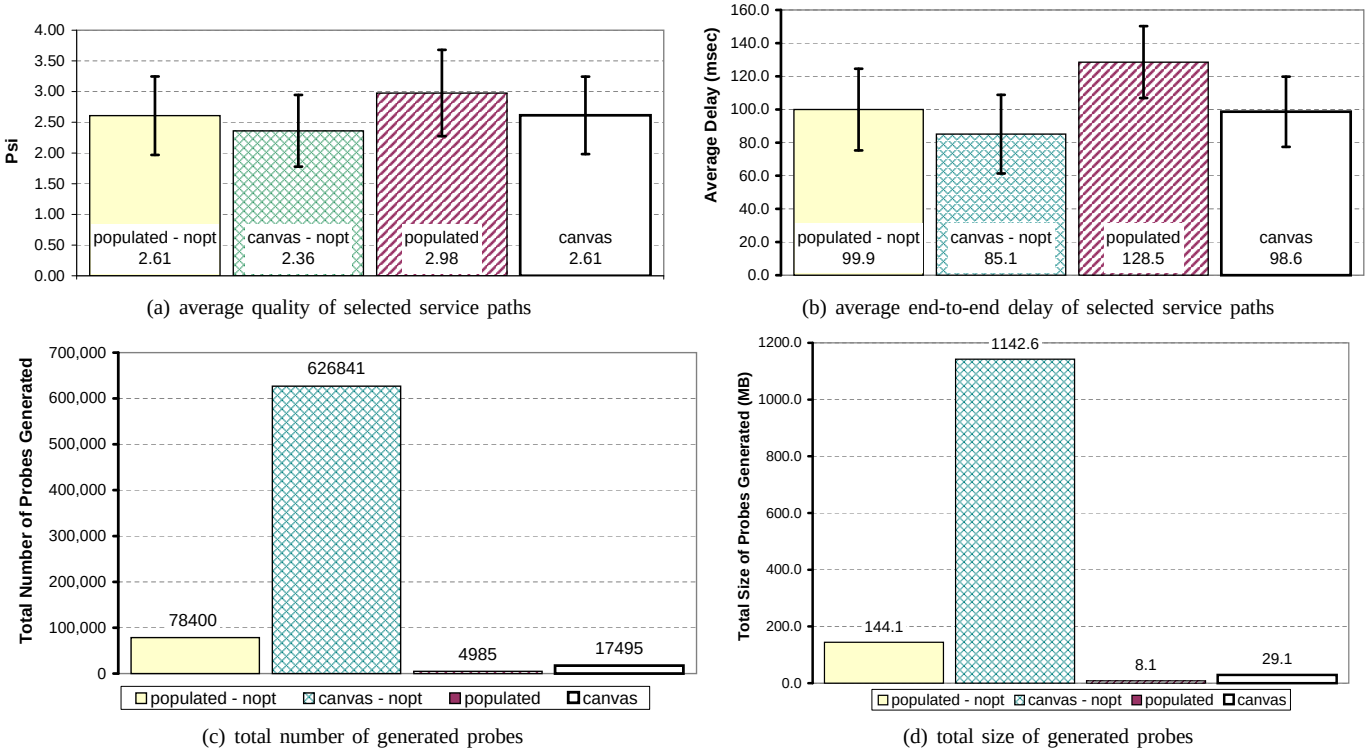
Notation	Meaning
T	buffer time-out
$\overline{H}$ upper bound	number of hops a probe can traverse
$\overline{W}$ upper bound	number of probes forwarded at each node
$\overline{B}$ upper bound	number of forwards for a probe at each node
$\underline{Q}$	expected quality improvement

V. Experimental Evaluation

We assess the performance of Dynamis in composing services on PlanetLab [18], an Internet research testbed comprising hundreds of Linux-based nodes distributed throughout the world. We have used the Service Clouds [14] prototype to apply Dynamis to different probing strategies. In these experiments we assume all service elements need to be executed in order (ordered service path). Due to space limitations, many experimental results are omitted here, but can be found in [17].

Test Setup and Procedure. Considering establishment of service paths between two nodes on the Internet, we first show that the proposed approach significantly reduces probing overhead by incorporating distributed selection (rather than selection at an endpoint), while still finding high-quality service paths. We use Formula 1 (Section IV) to measure the quality of a service path. Next, we evaluate the quality of the selected service paths in different approaches and configurations in terms of end-to-end delay. In particular, we assess the effect of $\underline{Q}$ and T parameters.

We measure the quality of the best service path found and the corresponding probing overhead. This implementation realizes a simplified Dynamis algorithm, which assumes two service elements of the same service path do not execute on the same overlay node (hence, probes are not forwarded to nodes



(a) average quality of selected service paths

(b) average end-to-end delay of selected service paths

(c) total number of generated probes

(d) total size of generated probes

Fig. 6. The quality and the end-to-end delay of selected services path vs. overhead of probing in the four different strategies.

they have already traversed). The test procedure cycles through source-destination pairs of nodes, submitting service requests to the system and allocating resources. We have implemented a simple protocol to reserve virtual resources. The source node performs the final service path selection and sends messages to reserve the required amount of resources on the overlay nodes along the selected service path.

Table III shows the settings of parameters. In case of multiple values for a parameter, the one in parentheses is used by default; unless stated otherwise. In these tests, all service specifications contain four ordered service elements, which consume the same amount of resources: 20 units of CPU processing power and 200Kbps of bandwidth. We conducted the experiments on 28 PlanetLab nodes, each assumed to have 100 units of CPU processing power and 1024Kbps uplink bandwidth. The results presented are the average of five samples of service composition between each pair of nodes.

TABLE III

Parameter	Value
distributed service selection	(enabled) , disabled
initial resource loads (CPU and bandwidth)	from (0%) to 60%
T	(500) , 1500 , 2000 msec
$\underline{Q}$	(0%) , 2% , 3% , 5% , 10%
$\overline{H}$ upper bound	N/A
$\overline{W}$ upper bound	(unlimited), 3000
$\overline{B}$ upper bound	unlimited
p_{s_i}	20 units of normalized CPU time
b_{s_j}	200 Kbps
D_{max}	300 msec
$\omega_p, \omega_b, \omega_d$ cofactors	1.0

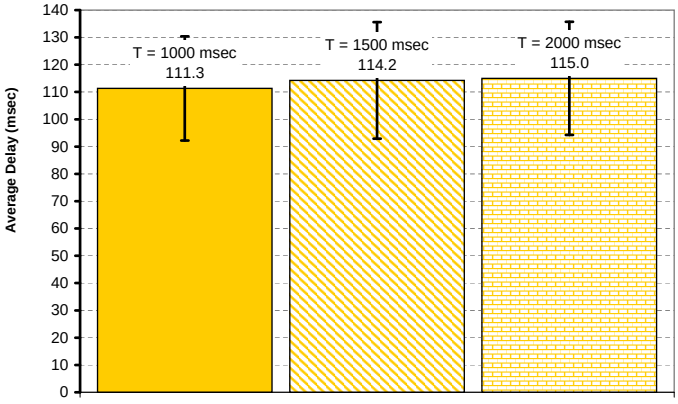
Results of Experiments. These tests compare four major strategies:

- *populated - nopt:* composes a service path using nodes that already host the required services; does not use distributed selection.
- *canvas - nopt:* considers the overlay as a blank computational canvas, so any service can be mapped to any node with sufficient resources; does not use distributed selection.
- *populated:* composes a service path using nodes that already host required services; uses distributed selection.
- *canvas:* considers the overlay as a blank computational canvas and uses distributed selection.

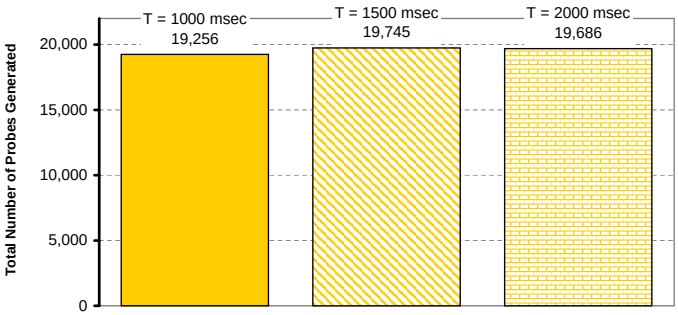
Figures 6(a) and 6(b) show the average quality and end-to-end delay for each of the strategies (95% confidence intervals included). While the values for the distributed selection cases ("populated" and "canvas") are slightly higher (therefore worse) than those for the first two cases, the differences are not significant. On the other hand, Figures 6(c) and 6(d) show that the probing overhead is dramatically reduced by distributed selection (over 93% in the " populated" strategy, and over 97% in the "canvas" strategy).

Figure 7 shows the effect of the parameter T in "canvas" strategy. These plots show no significant change as T is set to the different values. This observation is expected, since each node compares the quality of a partial service path both to all other ones received during the same epoch, and to those forwarded in previous epochs. Thus, unless the value of T is so small that few probes are received during an epoch, increasing the buffer timeout does not change the behavior of the strategies. Also, results from additional tests show that increasing $\underline{Q}$ has little effect on overhead, but does

increase end-to-end delay. We can conclude an epoch duration of $500msec$ is sufficient to significantly reduce the probing overhead of composing high-quality service paths, without the need for $\underline{Q}$, at least within the realm of these experiments.



(a) average delay of selected service paths



(b) total number of generated probes

Fig. 7. Effect of T on the service path delay and probing overhead ("canvas" strategy with Q = 5%).

VI. Conclusions and Future Work

We described Dynamis, a probing algorithm to support composition of distributed overlay services. Dynamis is based on distributed service selection, which reduces the overhead of probing to locate suitable nodes on which to instantiate services. We presented the Dynamis algorithm and used it to empirically assess performance of different service composition strategies on the Internet. The experimental results show that using distributed selection reduces the probing overhead in service composition, while still finding high-quality service paths. Future work can include experiments with fewer assumptions, such as evaluation within a heterogeneous assortment of nodes with different capabilities and resources; as well as addressing quality of service in terms of non-functional requirements, such as trustworthiness and reliability. Furthermore, our ongoing research addresses the design of an autonomic framework to compose adaptive distributed stream processing services, including real-time data streams generated by sensor networks that monitor ecosystems.

Further Information. Related publications and software on the RAPIDware project can be found at the following website: `http://www.cse.msu.edu/rapidware`.

References

[1] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, (Washington, DC, USA), pp. 49–60, IEEE Computer Society, 2006.

[2] T. Repantis, X. Gu, and V. Kalogeraki, "Synergy: Sharing-aware component composition for distributed stream processing systems," in *Proceedings of the 7th ACM/IFIP/USENIX International Middleware Conference (MIDDLEWARE 2006)*, vol. 4290 of *LNCS*, (Melbourne, Australia), Springer-Verlag, November 2006.

[3] S. D. Gribble, M. Welsh, J. R. von Behren, E. A. Brewer, D. E. Culler, N. Borisov, S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao, "The Ninja architecture for robust Internet-scale systems and services," *Computer Networks*, vol. 35, no. 4, pp. 473–497, 2001.

[4] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *IEEE Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[5] P. Grace, G. Coulson, G. Blair, L. Mathy, D. Duce, C. Cooper, W. K. Yeung, and W. Cai, "GRIDKIT: Pluggable overlay networks for Grid computing," in *Proceedings of International Symposium on Distributed Objects and Applications (DOA)*, (Larnaca, Cyprus), pp. 1463–1481, October 2004.

[6] J. Xiao and R. Boutaba, "QoS-aware service composition and adaptation in autonomic communication," *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 2344–2360, December 2005.

[7] X. Fu and V. Karamcheti, "Automatic creation and reconfiguration of network-aware service access paths.," *Computer Communications*, vol. 28, no. 6, pp. 591–608, 2005.

[8] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan, "Resource-aware distributed stream management using dynamic overlays," in *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS 2005)*, (Columbus, OH, USA), pp. 783–792, IEEE Computer Society, June 2005.

[9] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, (Asilomar, CA, USA), pp. 277–289, January 2005.

[10] X. Gu, K. Nahrstedt, and B. Yu, "SpiderNet: An integrated peer-to-peer service composition framework," in *Proceedings of IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*, (Honolulu, Hawaii), pp. 110–119, June 2004.

[11] J. Liang and K. Nahrstedt, "Service composition for advanced multimedia applications," in *Proceedings of 12th Annual Multimedia Computing and Networking (MMCN'05)*, vol. 5680, (San Jose, California), pp. 228–240, January 2005.

[12] B. J. Bonfils and P. Bonnet, "Adaptive and decentralized operator placement for in-network query processing.," *Telecommunication Systems*, vol. 26, no. 2-4, pp. 389–409, 2004.

[13] S. Seshadri, V. Kumar, and B. F. Cooper, "Optimizing multiple queries in distributed data stream systems," in *Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW)*, (Atlanta, GA, USA), pp. 25–30, IEEE Computer Society, April 2006.

[14] F. A. Samimi, P. K. McKinley, S. M. Sadjadi, C. Tang, J. K. Shapiro, and Z. Zhou, "Service Clouds: Distributed infrastructure for adaptive communication services," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 4, pp. 84–95, September 2007.

[15] C. Tang and P. K. McKinley, "Improving mutipath reliability in topology-aware overlay networks," in *Proceedings of the Fourth International Workshop on Assurance in Distributed Systems and Networks (ADSN), held in conjunction with the 25th IEEE International Conference on Distributed Computing Systems*, (Columbus, Ohio), June 2005.

[16] G. Brassard and P. Bratley, *Fundamentals of Algorithmics*. Prentice Hall, 1996.

[17] F. A. Samimi and P. K. McKinley, "Dynamis: Dynamic overlay service composition for distributed stream processing," Tech. Rep. MSU-CSE-06-39, Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, December 2006. Available at `http://www.cse.msu.edu/~farshad/serviceclouds`.

[18] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proceedings of ACM Workshop on Hot Topics in Networks*, (Princeton, New Jersey), pp. 59–64, October 2002. `http://www.planet-lab.org/`.