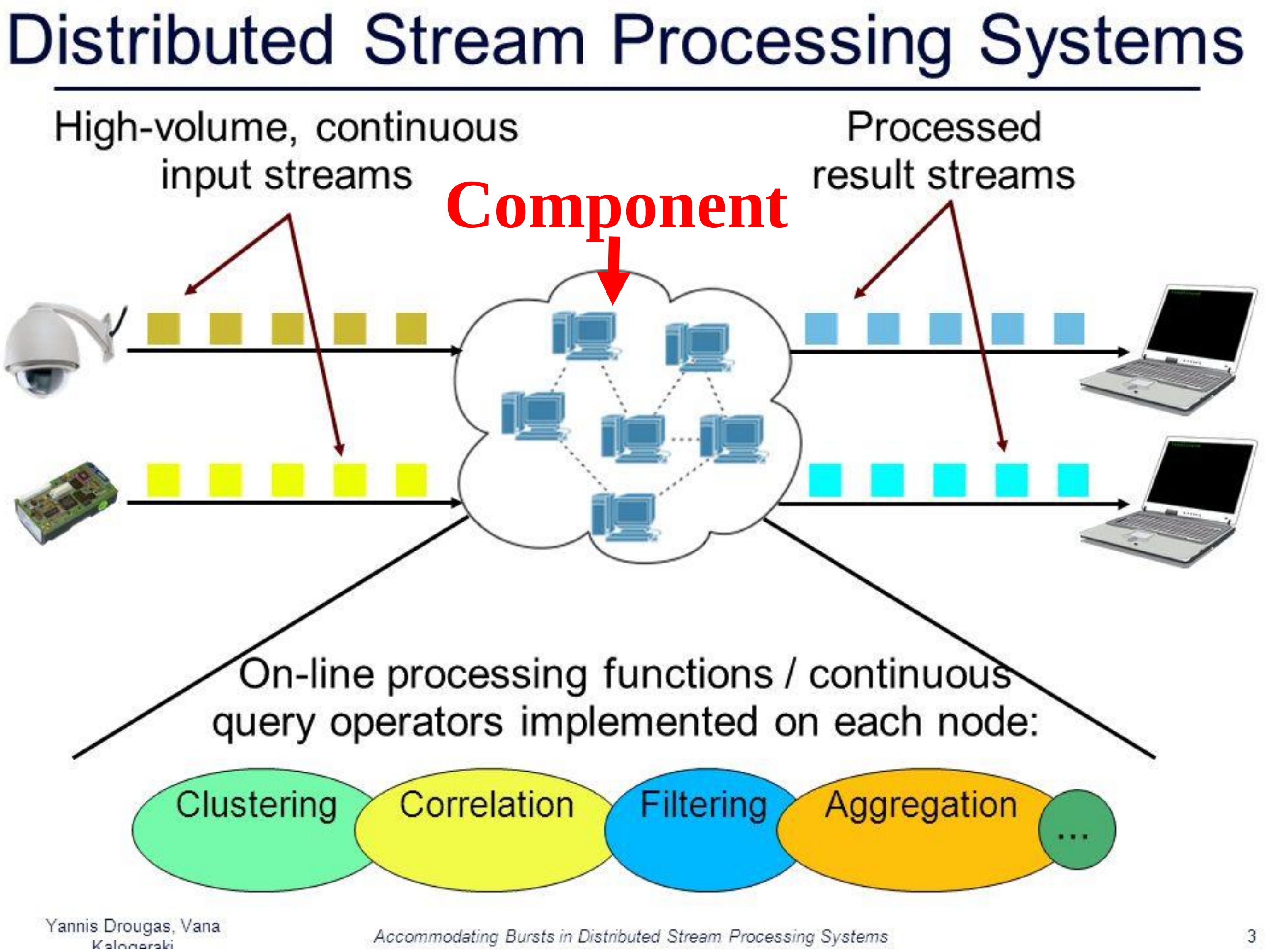
QoS-Aware Shared Component Composition for Distributed Stream Processing Systems

Thomas Repantis, Member, IEEE, Xiaohui Gu, Member, IEEE, and

Vana Kalogeraki, Member, IEEE

IEEE Transactions on Parallel and Distributed Systems, 20(7), 968-982

Introduction - Distributed Stream Processing Systems



Distributed Stream Processing Systems

High-volume, continuous input streams

Component

Processed result streams

On-line processing functions / continuous query operators implemented on each node:

Clustering

Correlation

Filtering

Aggregation

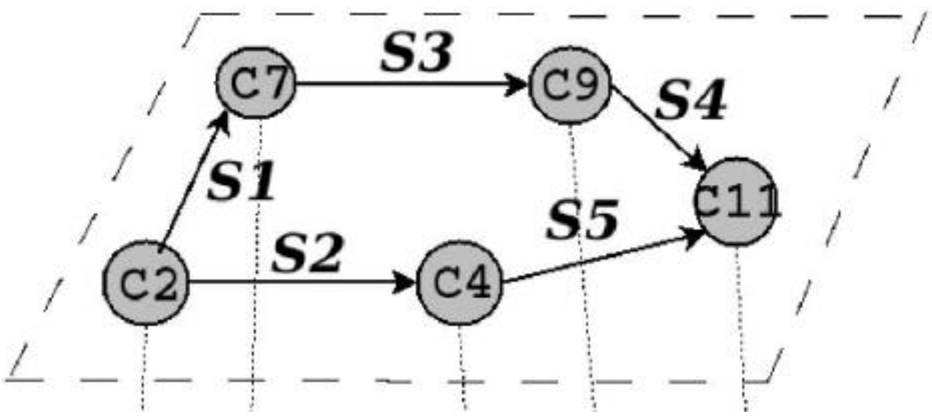
...

Yannis Drougas, Vana Kalogeraki

Accommodating Bursts in Distributed Stream Processing Systems

3

Distributed Stream Processing Application



Motivation

- In **DSPS**s (**D**istributed **S**tream **P**rocessing **S**ystems),

  streams continuously arrive components,

  components need to process input streams in real time to generate output streams.

Major challenge:

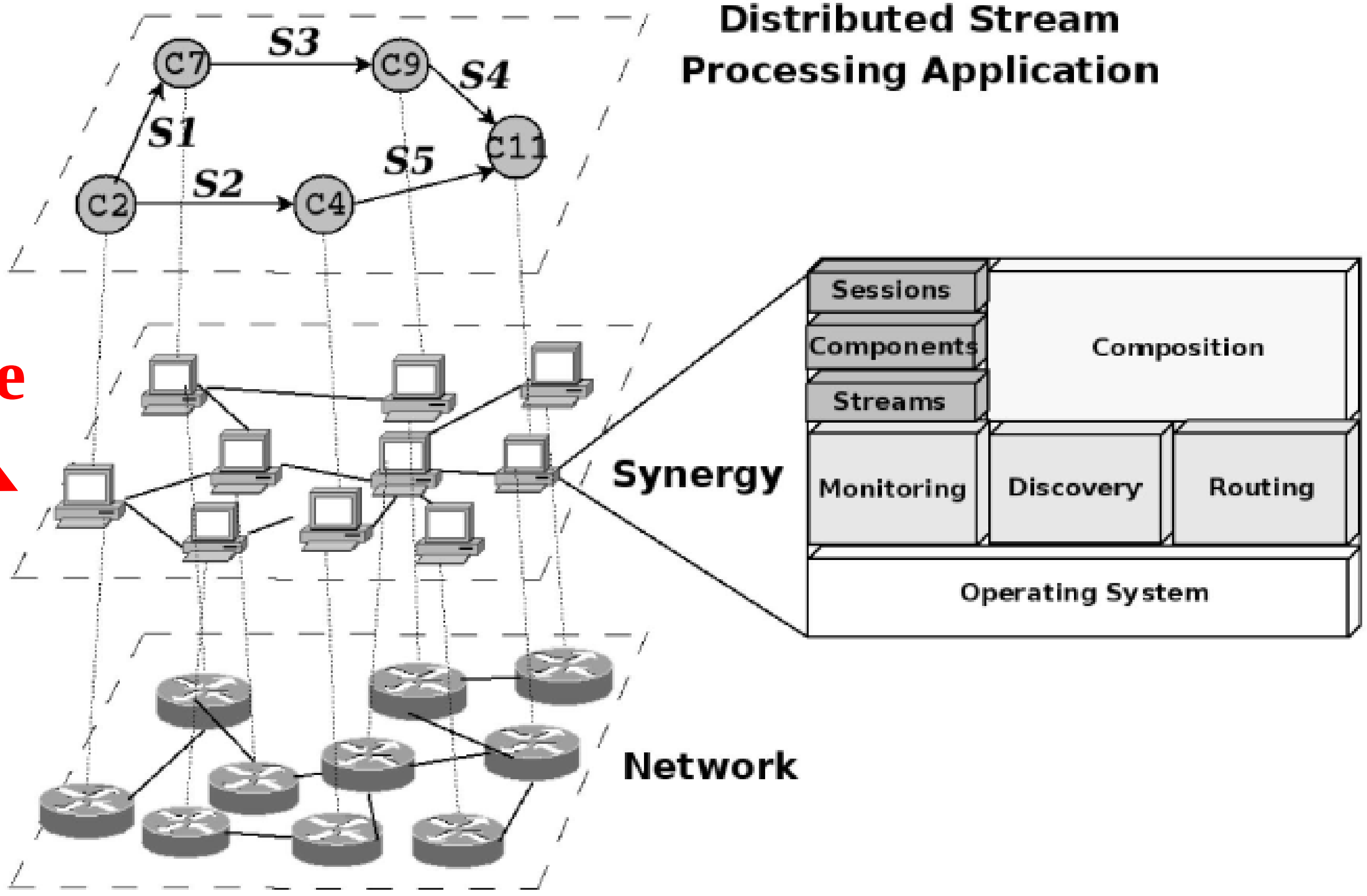
Select among different component to compose stream processing applications on demand.

Motivation

- focuses on enabling sharing-aware component composition for efficient distributed stream processing.

- Sharing-aware composition allows different applications to utilize:
  - previously generated streams
  - already deployed stream processing components

System model - **Synergy Architecture**



Middleware

Fig. 2. Synergy system architecture.

Algorithm - **Synergy component composition protocol**

Input: query $\langle \xi, Q_{\xi}, \rangle$, node v_s

Output: application component graph λ

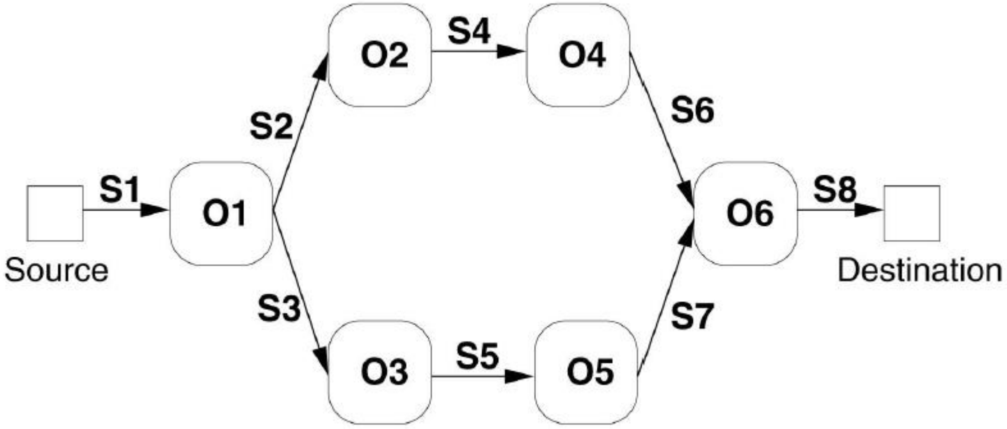
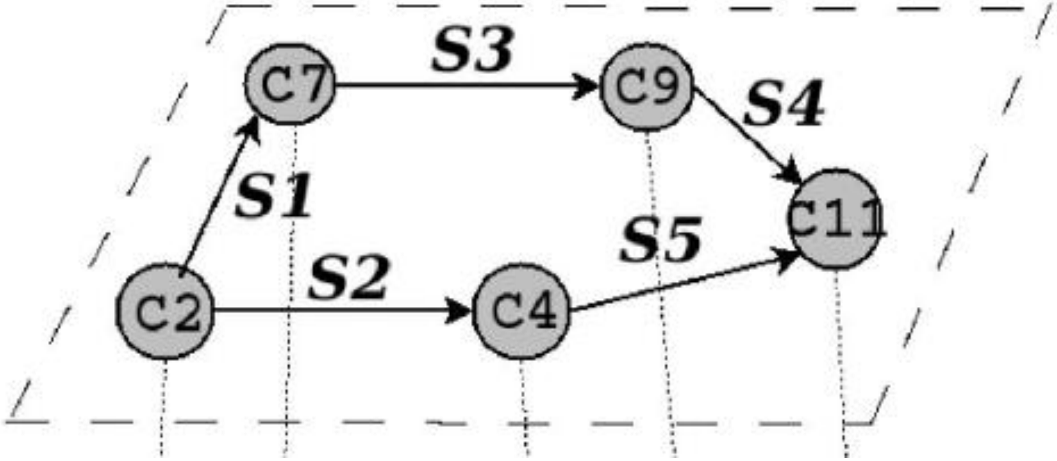


Fig. 5. Query plan example.



Algorithm - **Synergy component composition protocol**

1 v_s identifies *maximum sharable point(s)* in ξ
2 v_s spawns initial probes

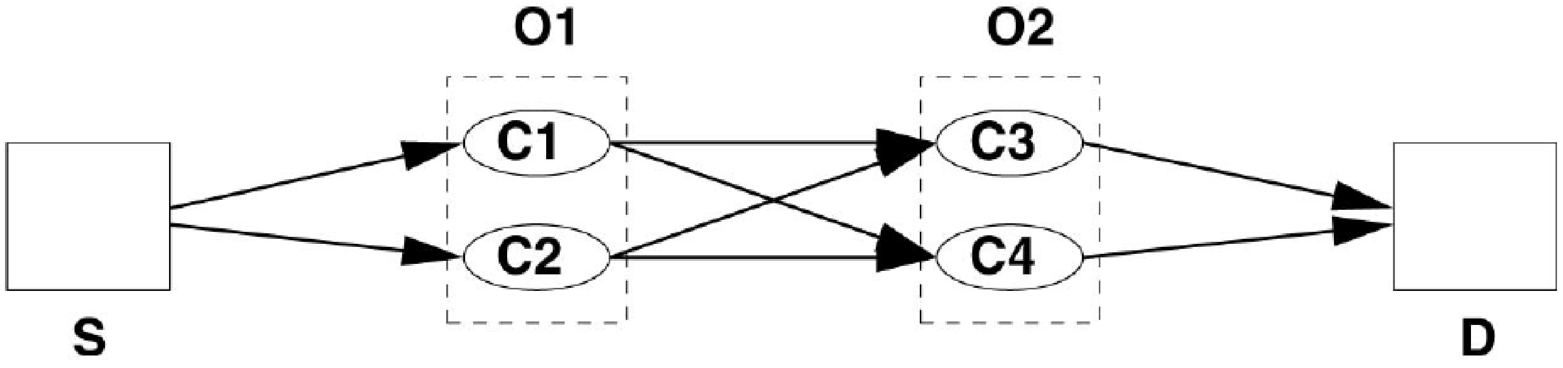


Fig. 3. Probing example.

Probing path: 1. S -> C1 -> C3 -> D
2. S -> C1 -> C4 -> D
3. S -> C2 -> C3 -> D
4. S -> C2 -> C4 -> D

Algorithm - **Synergy component composition protocol**

```
for each $v_i$ in path
    checks available resources
    AND checks QoS so far in $Q_\xi$
    AND checks projected QoS impact
```

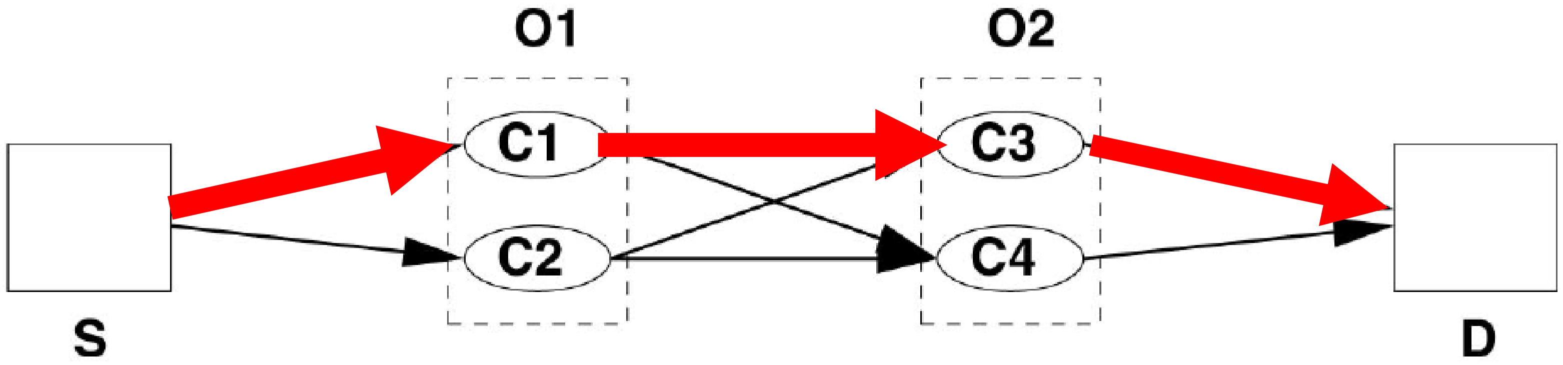
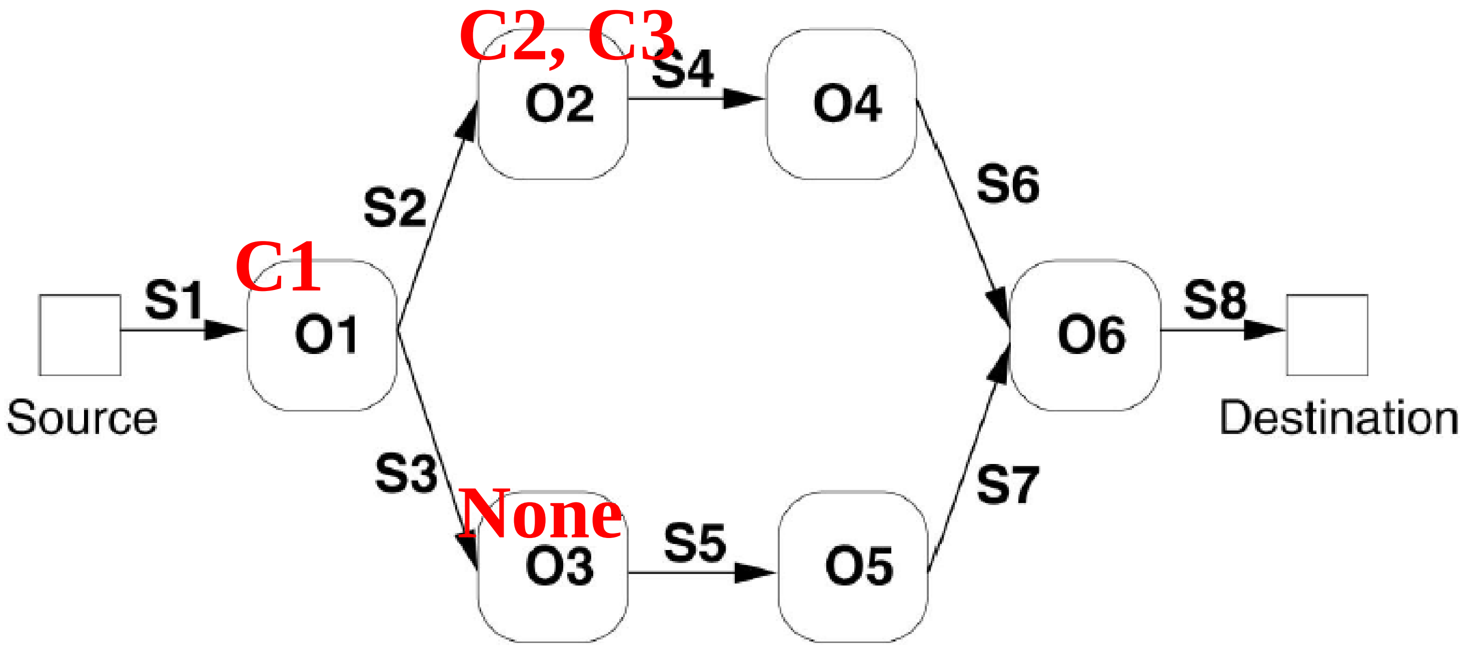


Fig. 3. Probing example.

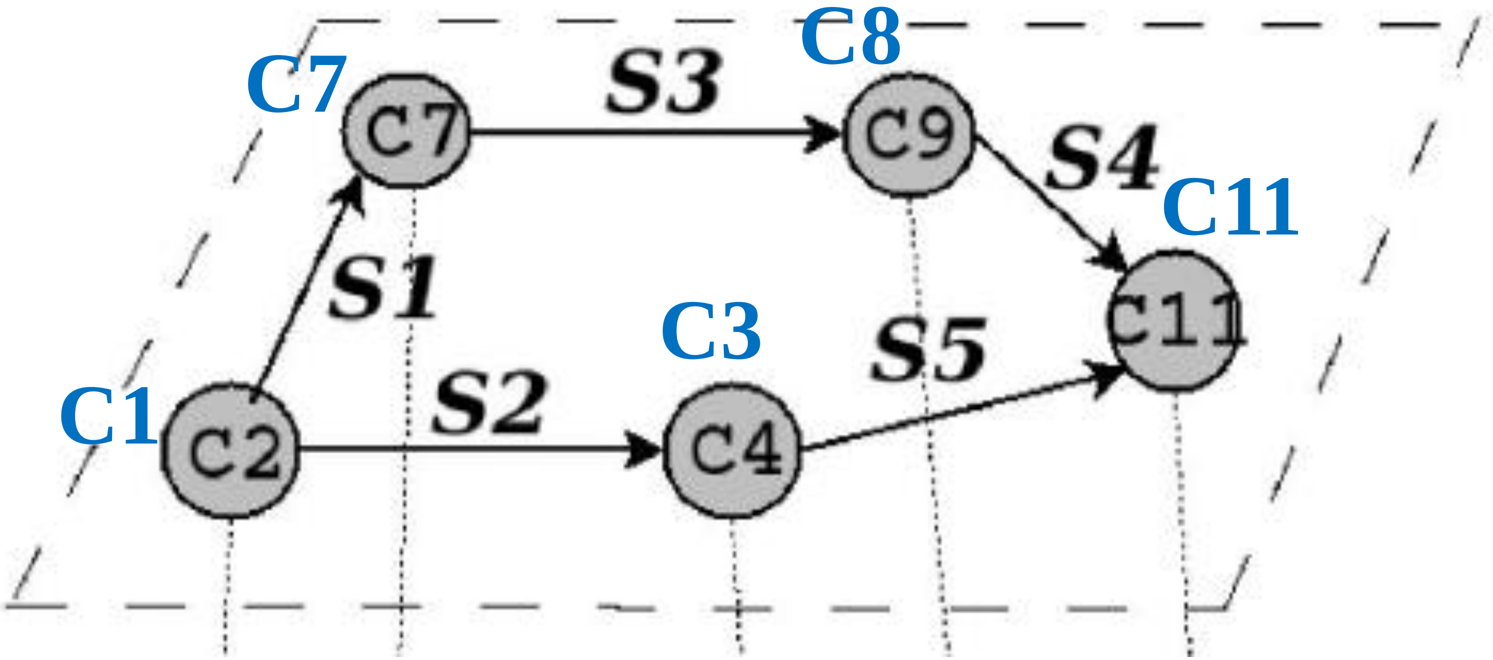
Algorithm - **Synergy component composition protocol**

```
if probed composition qualifies
    sends acknowledgement message to upstream node
    performs transient resource reservation at $v_i$
    discovers next-hop candidate components from $\xi$
    deploys next-hop candidate components if needed
    spawns probes for selected components
else drops received probe
```



Algorithm - **Synergy component composition protocol**

```
14 $v_s$ selects most load-balanced component composition $\lambda$
15 $v_s$ establishes stream processing session
```



Experimental Evaluation - **Setup**

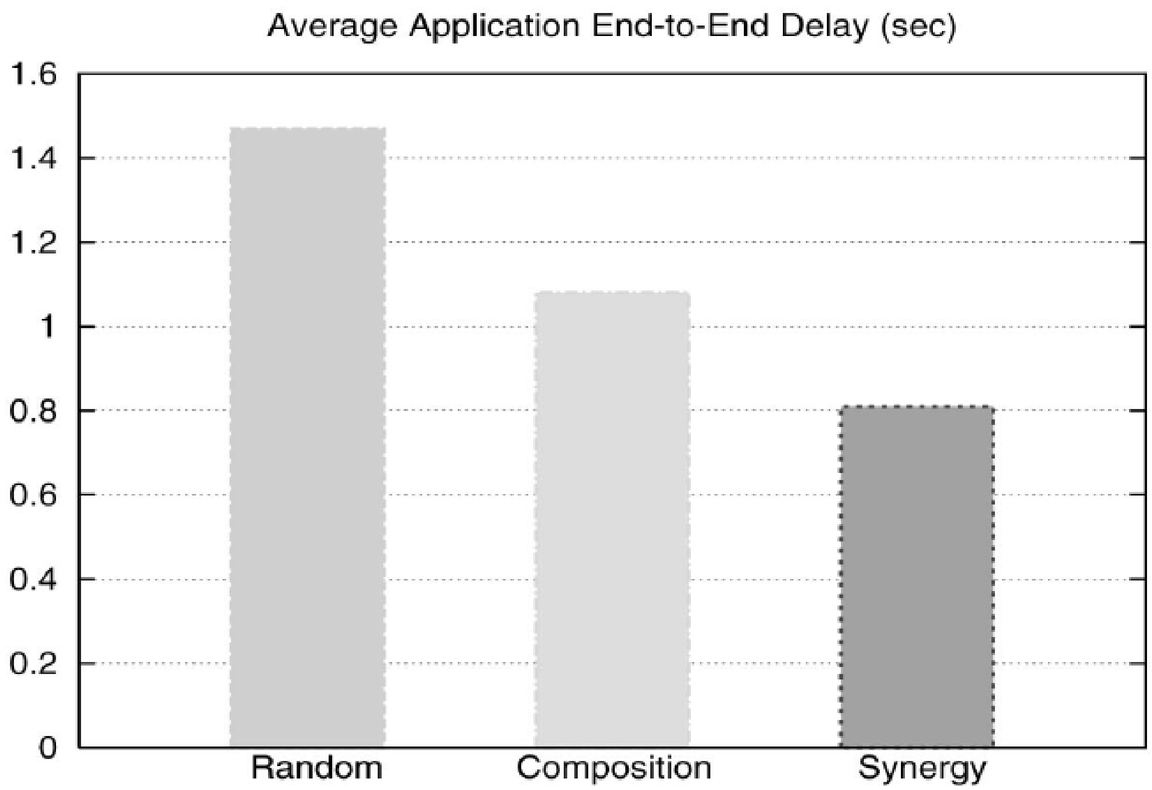
- Implemented as a multithreaded system including about 20,000 lines of Java code
- Running on each of 88 physical nodes of PlanetLab.
- Based on the SpiderNet service composition framework.
- One hundred components were deployed uniformly across the nodes, with a replication degree of 5.
- Application requests asked for two to four components chosen randomly and for the corresponding streams between the components.
- Generate approximately nine requests per second throughout the system, using a Zipf distribution with $\alpha = 1.6$

Experimental Evaluation - **Setup**

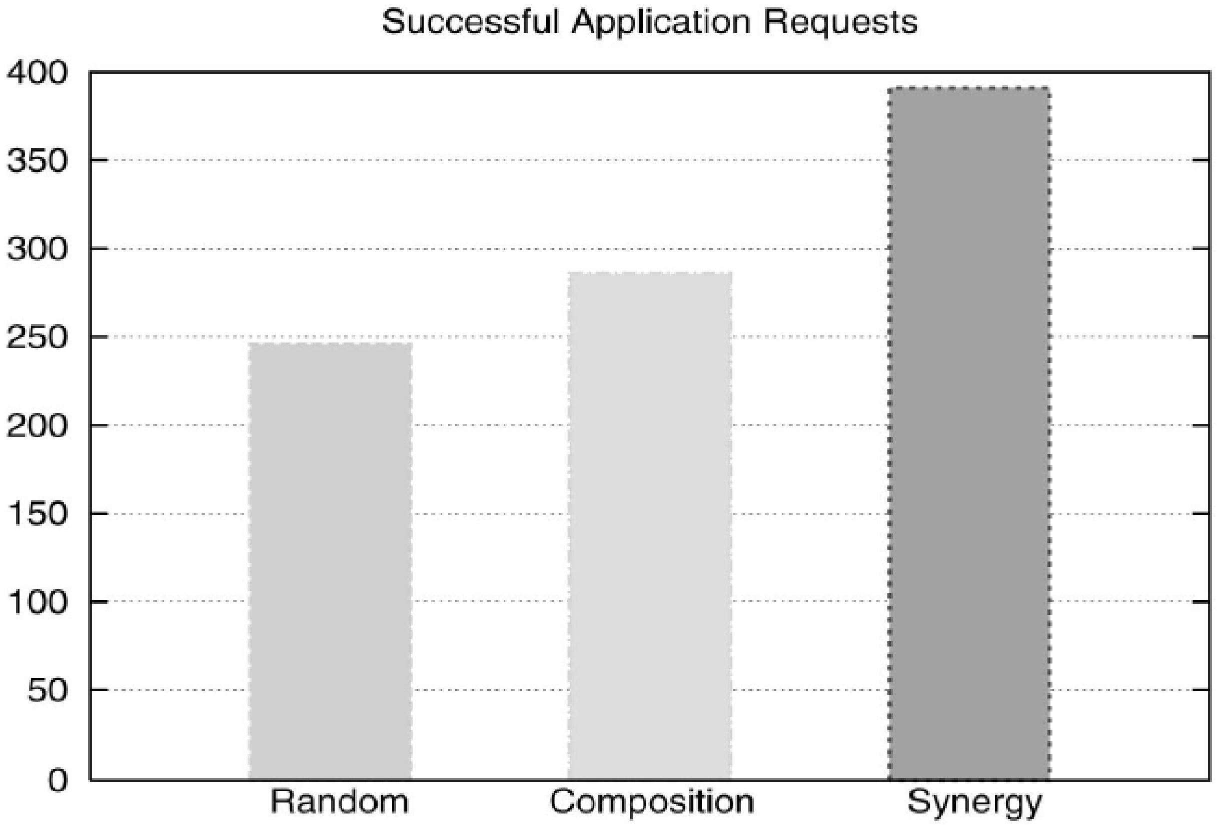
- **Compared Synergy against two different composition algorithms:**
  1. A Random algorithm that randomly selected one of the candidates for each application component
  2. a Composition algorithm performs QoS-aware composition but does not consider result stream reuse or component reuse

Experimental Evaluation

Average application end-to-end delay

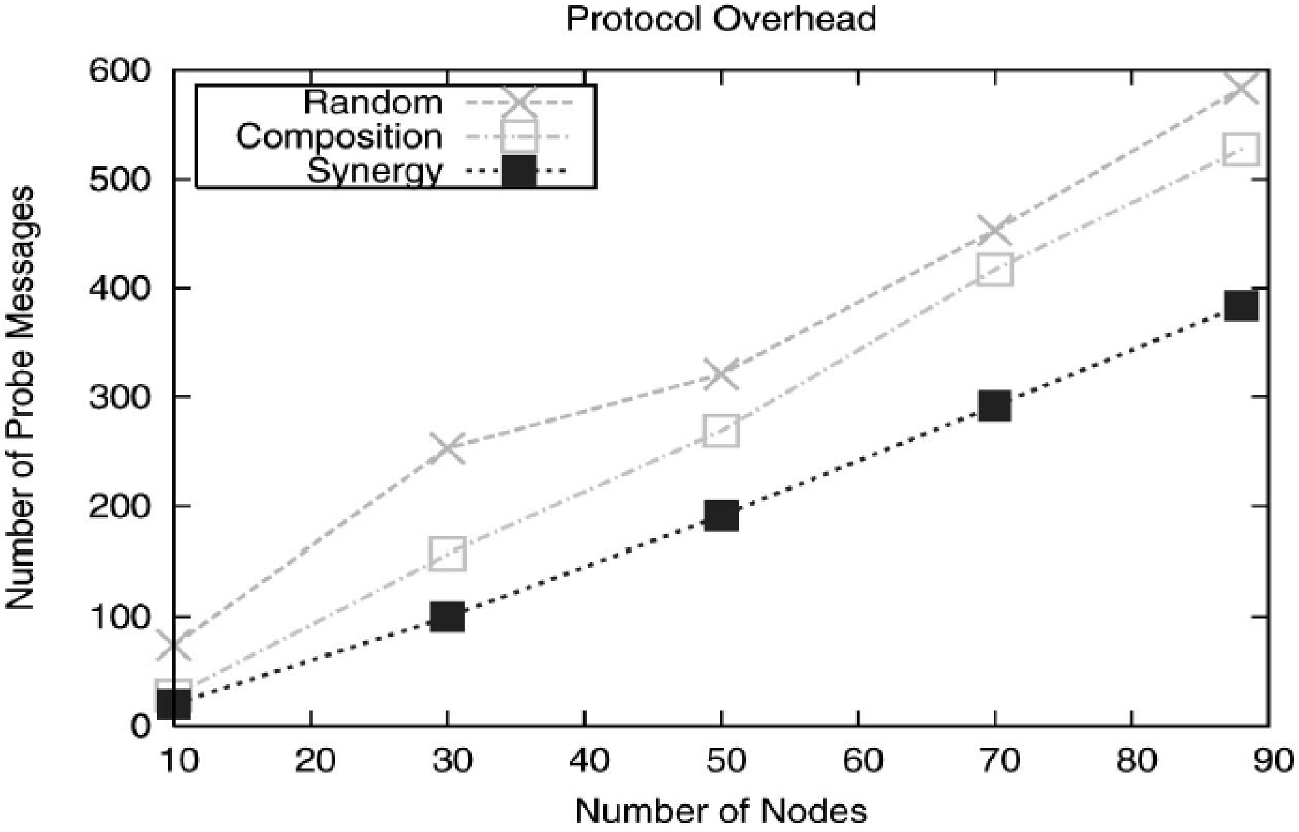


Successful application requests

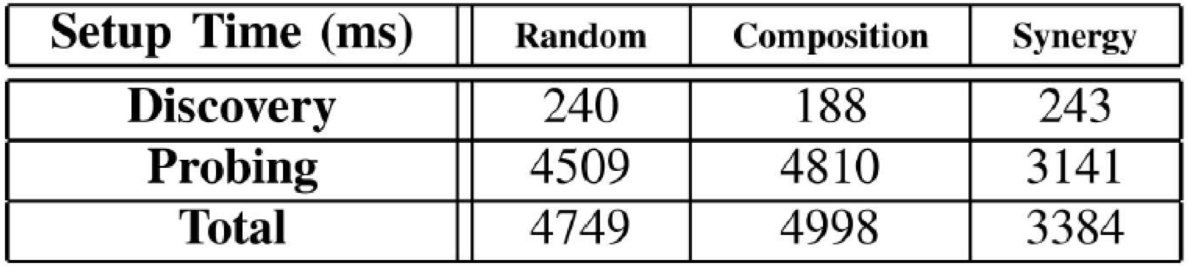


Experimental Evaluation

Protocol overhead



Breakdown of average setup time



Setup Time (ms)	Random	Composition	Synergy
Discovery	240	188	243
Probing	4509	4810	3141
Total	4749	4998	3384

Conclusions

- Synergy:
  - built on top of a totally decentralized overlay architecture
  - reuse existing streams and components
  - ensure that the QoS requirements of the currently running applications

- Prototype implementation of Synergy over PlanetLab shows that:
  - sharing-aware component composition can enhance QoS provisioning for distributed stream processing applications.