Synchronising service compositions in dynamic ad hoc environments

Christin Groba and Siobhán Clarke
Lero Graduate School of Software Engineering
Distributed Systems Group
School of Computer Science and Statistics
Trinity College Dublin, Ireland
grobac, Siobhan.Clarke@scss.tcd.ie

***Abstract*—Service compositions in dynamic ad hoc environments face frequent changes in the network and service topology. Late service binding is a way to adapt to runtime changes but requires additional communication over error-prone and energy-constrained networks. In particular, synchronising parallel service flows relies on steady message exchange when mutually unknown service providers have to agree on a common merge node. Binding the merge node early through a single entity reduces the need for coordination. However, it compromises flexibility and may nonetheless increase communication as the decision entity has only partial view of the system. This paper proposes a multi-party late binding protocol as a solution for flexible, communication-aware service composition in dynamic ad hoc environments. The protocol integrates the discovery of unknown synchronisation partners with the exchange of binding suggestions to reduce communication. We embed this protocol in our model for opportunistic service composition and evaluate it against an early binding approach. Simulation results show that while being overall equally successful, late service binding outperforms early binding with respect to communication overhead and response time.**

***Keywords*-service-composition; AND-split; AND-join; mobile; ad hoc; dynamic binding; peer-to-peer;**

I. Introduction

The composition of existing services for new value-added functionality has been widely used to manage inner and inter-organisational processes on the Web. Technological advances such as embedding sensors in everyday objects and the ability to communicate among devices create new opportunities for service-oriented applications. The following scenario is set in the context of mobile phone sensing [1] and motivates service composition as a model for collaboration between smart devices.

In the scenario, microphone sensors on mobile phones track their owners' daily exposure to noise. The process involves sampling, filtering, classifying, and geo-tagging audio data. However, the phone is primarily likely to be in the owner's pocket and unable to record high-quality data. The classifier software on the phone is unfamiliar with the current environment and not properly trained to categorise the noise. Further, the GPS function required for geo-tagging is temporarily unavailable. An innovative way of solving these issues is by collaborating with nearby mobile phones that share their audio capabilities [2]. The interaction among multiple devices can be modelled as a service composition and is required if no single device has enough capabilities to complete the complex task alone. In the scenario, sampling from three unique devices improves data quality. The sampling services then forward their results to the same aggregation service which hands on to a well-trained classifier and further on to a geo-tagging service (Figure 1). The devices hosting these services are in close proximity and communicate via an ad hoc network.

In contrast to traditional managed and resource-abundant environments, service compositions in dynamic ad hoc networks face frequent topology changes, scarce energy resources, and limited network bandwidth. Late service binding adapts to these runtime changes as it selects service providers only prior to their invocation. However, with an increase in the request complexity, coordination and communication over error-prone network links increases. Requests optimised for data quality and latency introduce parallel execution paths that need to synchronise in a common merge node. Late binding of such service flows involves two challenges [3]: First, multiple service providers have to agree on the same merge provider at runtime. Second, synchronisation partners are mutually unknown as they, too, are bound late. Getting to know each other and finding an agreement requires a number of messages whose delivery may fail due to network overload and node mobility. Subsequent failure recovery produces even more communication that strains the limited energy resources in the network.

Research has led to composition solutions that bind a merge node early through a single entity [4], [5], [6]. Although this reduces the coordination effort, it also reduces the composite's flexibility to react to runtime changes. Further, a single decision entity cannot anticipate the actual communication cost between the selected merge node and its yet-to-be-bound predecessors. Late binding solutions have a more up-to-date system view but require a number of messages to be exchanged to mutually introduce synchronisation partners [3].

This paper proposes a multi-party late binding protocol to address the challenges of synchronising service compositions in dynamic ad hoc environments. The protocol binds a common merge node only prior to its execution

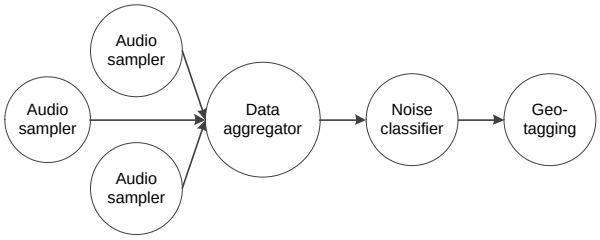


Figure 1. Service request for tracking exposure to noise

and preserves the composite's ability to react on runtime changes. Synchronisation partners discover each other in an ad hoc manner and include their local binding suggestions in the discovery message to reduce communication. The final selection considers the input of all partners and allows for distributed cost considerations.

The contribution of this paper is a flexible communication-aware protocol for synchronisation partners unknown to each other to achieve late binding to a common merge node at runtime. The protocol addresses limitations of earlier work [7] and adds synchronising parallel service flows to our model for opportunistic service composition. This improves the understanding of how complex tasks can be coordinated in dynamic ad hoc networks using service composition.

The paper is organised as follows: Section II outlines existing solutions for handling synchronising service compositions. Section III describes the proposed protocol within the opportunistic composition model. Section IV explains the simulation setup used to evaluate the protocol against an existing solution in an inner-city environment. Section V presents the results of the simulation study. Section VI summarises the protocol and discusses its trade-offs.

II. Related work

Research in dynamic service composition has produced a number of strategies to bind and invoke service providers at runtime and to manage synchronising service flows.

Broker-based designs employ a single coordinator that centrally binds and invokes services [8], [9]. A broker synchronises parallel service flows naturally without additional effort because it executes the composition logic locally. However, service providers do not communicate directly and rather send their invocation results back to the broker. This star-based interaction increases communication.

Fragmentation-based solutions split request descriptions into individual activities with corresponding composition logic using executable workflow networks [10], dependency tables [11], or chemical reaction rules [12]. The fragments for the parallel execution paths include the address of the common merge node and avoid further negotiation during execution [4]. However, all services are bound before the overall execution can start. This wastes scarce resources for conditional or prematurely-terminated paths and is less flexible towards runtime changes that render selections invalid and cause failure. Dynamic leader election in logical peer groups [13] adapts to dynamically arriving and departing providers but has been proposed only for sequential requests. OSIRIS [5] makes its pre-selection final only during execution, but requires a reliable and globally available synchronisation node to handle parallel service flows. In dynamic ad hoc networks such a node may not exist.

Decentralised allocation algorithms either probe the network for multiple high-quality execution paths or assign providers incrementally hop-by-hop. Probe-based algorithms promote early binding similar to fragmentation-based designs because they either require the selection to complete prior to execution [14] or postpone final best path selection to the destination [15], [16], [17]. Existing hop-by-hop algorithms would allow for late binding. However, they are not readily applicable because they are either constrained to sequential requests [18] or require the destination node to first assemble parallel paths before execution can start [19].

Continuation passing [6] binds a merge node together with the corresponding split node and forwards the merge address with the remaining request description. This way, a single entity can make the selection decision during execution without additional runtime agreement. However, the system view of such an entity is limited. It cannot foresee which yet-to-be-bound service providers will actually interact with the merge node and how well connected they are.

Dynamic synchronisation strategies [3], similarly to this paper, address the problem of binding a merge service late by giving the responsibility to its immediate predecessors. A parallel path updates all other paths on each service allocation it makes. This way, final synchronisation partners in parallel paths know each other and can run an agreement protocol. Such a strategy requires many messages to be exchanged and strains the scarce network resources.

Election and consensus solutions for distributed systems relate to some extent to the late binding problem. Election protocols have to ensure that all nodes in the network refer to the same new coordinator after the current one has disconnected [20], [21]. The election is led by a single decision maker who needs global system view to consider the logical distance between each node and the new coordinator in its selection process [22]. Maintaining such a view is expensive and infeasible in true peer-to-peer networks [15]. Further, late binding involves only a subset of nodes which can communicate in a more targeted way than through controlled network flooding [20], [21]. Consensus protocols achieve an agreement among agents if the opinion of all agents stabilises. Stochastic consensus models [23] assume that agents are connected to each other and aware of all possible opinions. The protocol proposed in this paper explores a communication-efficient way to establish such knowledge by mutually introducing synchronisation partners and revealing possible merge candidates.

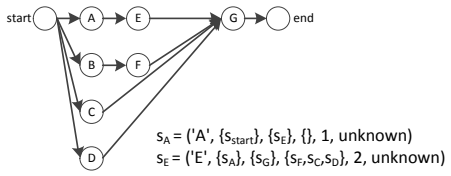


Figure 2. Example for complex service requests

III. Multi-party late service binding

Multi-party late service binding addresses the need for a flexible communication-aware composition strategy that satisfies complex service requests in ad hoc networks with volatile links between energy-constrained service providers.

A. System model

The input for a composition is an abstract request description that defines the nature and order of required services. At runtime the composition incrementally allocates suitable providers to make the description executable and invokes it to return a final composition result. This paper focuses on requests that can be modelled as a well-structured[1] directed graph $G = (S, A)$ with AND-splits and AND-joins. S represents the set of services and A the set of control arcs. The functions $src : A \rightarrow S$ and $dst : A \rightarrow S$ return the unique source and destination service of an arc. A service s has a set of input dependencies $I = \{s_i | \exists a \in A, src(a) = s_i \wedge dst(a) = s\}$ and a set of output dependencies $O = \{s_i | \exists a \in A, src(a) = s \wedge dst(a) = s_i\}$. The underlying control logic is modelled implicitly and corresponds to AND-logic for multiple input or output dependencies of a service. A service q has a set of synchronisation partners $P = \{p_i | p_i, q \in I_s \wedge \neg(p_i = q)\}$ which includes all services that are in addition to q input dependencies of a merge service s. The graph has a unique start service for which I is empty and a unique end service for which O is empty. The depth $dpth$ of a service is the length of its longest path back to the start service. Further, a service has a unique name and a provider address which is initially unknown. A service can be described as $s = (name, I, O, P, dpth, adr)$. Figure 2 shows an example for a complex service request and two descriptions for constituent services.

The operating environment consists of a set of autonomous mobile nodes that have the capability to host services, handle synchronisation logic, and forward messages. These nodes are initially unaware of each other and establish network connections only on request. Nodes that participate in a composition interact with each other in a fully decentralised peer-to-peer manner.

[1]Polyvyanyy et al. [24] describe well-structured as follows: " ... for every node with multiple outgoing arcs (a split) there is a corresponding responding node with multiple incoming arcs (a join), such that the set of nodes between the split and the join form a single-entry-single-exit region".

B. Overview

Multi-party late service binding is embedded in an opportunistic composition model for wireless ad hoc networks. The model combines on-demand service discovery with situational advertising, hop-by-hop selection and immediate invocation of late bound service providers.

The state-transition diagram in Figure 3 shows the model from a node's perspective and outlines how the node processes a composition request. The node starts engaging in a composition, when it receives a request description from another node for which it provides the next required service. It stores the request for later dependency analysis and responds by sending back an application message that indicates the node's capabilities to execute the next required service. The node is accepted, if it receives a token that identifies it as the allocated service provider. A token contains the service input and represents the transfer of binding control. Once the node has received a token from all input dependencies of the allocated service, it executes the service. Afterwards, it extracts the service's output dependencies and synchronisation partners from the request and searches its local repository for corresponding provider addresses. The repository maintains a service-to-provider mapping for the composition that the node is currently engaged in and updates its knowledge each time it overhears a message. If the node can bind all output dependencies to provider addresses and the set of synchronisation partners is empty, it creates tokens and hands them over to its successors. If it fails to allocate output dependencies, the node searches for potential providers. In case synchronisation is required, the node transitions from the *selecting* to the *syncing* state, sends its partners a *sync* message, and waits for the same from them. A *sync* message identifies the sender as a partner service and contains a suggestion for the next provider. Once the node has received a *sync* message from all its partners, it makes its final selection decision. If at least one merge provider has been proposed, the node can select one and hand it over its token. Otherwise, the node searches for a provider and synchronises with its partners again.

The arrangement of the *executing, selecting,* and *handover* states illustrates late binding. The current token holder makes the binding decision only after it has executed. This differs from approaches that start decentralised execution only after all services have been bound.

C. Synchronised late binding

Parallel execution paths do not explicitly exchange their binding decisions for intermediate services and a node may not be in range to overhear binding messages. If the local repository fails to return partner addresses, the node needs to discover them in an ad hoc manner. Searching for partners involves some kind of controlled network flooding and implies a high communication overhead. The protocol uses

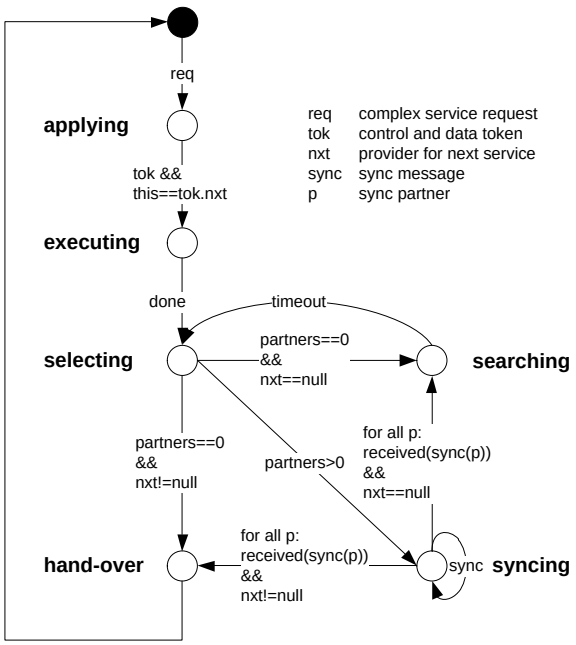


Figure 3. Opportunistic composition with synchronised late binding

Input: s, d {source and destination service}
Output: hop {address of next hop}

1: Queue q, Service n
2: $q.add(d)$
3: **while** q not empty **do**
4: $n \leftarrow q.dequeue()$
5: **if** $n.adr$ known **or** $n.dpth < min(s.dpth, d.dpth)$ **then**
6: break
7: **else**
8: **for all** i in $n.I$ **do**
9: $q.uniqueAdd(i)$
10: **end for**
11: **end if**
12: **end while**
13: $hop \leftarrow n.adr$
14: **if** hop unknown **then**
15: $i \leftarrow pickone(s.I)$
16: $hop \leftarrow i.adr$
17: **end if**
18: **return** hop

Figure 4. Service layer routing to synchronise with unknown partner

service layer routing instead to deliver a *sync* message to an unknown partner address.

The proposed service layer routing algorithm determines the next hop from a source service to a destination service for a given composition request (Figure 4). Starting from the destination service the algorithm checks whether the source repository knows the corresponding provider address (Line 2-5). If this is not the case, the destination's input dependencies are checked (Line 8). The breadth-first backward traversal of input dependencies stops if an address has been found or all services between the source and destination depth have been checked (Line 5). If the next hop is unknown, the address of one of the source's input dependencies is returned (Line 14-17). This means if routing to another path does not succeed, the a message will be routed backwards in source's path.

In the late binding protocol, the original *sync* message sender executes the routing algorithm with its allocated service as source and a partner service for which the provider address is unknown as destination. It sends the *sync* message to the address that the routing algorithm returns. Non-partner nodes that receive *sync* messages run the algorithm again with their own allocated service as source and the original partner as destination. Backward routing stops at the latest with the split node because it bound the first service of each parallel path and thus knows their addresses. Once in the partner path, service providers know the addresses of their output dependencies and route forward to the destination (Line 5 first condition). In case a *sync* message arrives in the partner path before the partner is bound, the message is stored and dispatched with each next binding step.

The final selection occurs when each partner has received a *sync* message from all other partners. Each partner then creates a map that lists the potential next providers and their selection criteria (e.g., number of hops from the proposing partner to the proposed provider). Applying the same selection algorithm on the same map, the partners individually come to the same conclusion of which node to allocate to the merge service. The current selection algorithm prioritises the most common provider over the one with the best individual criteria. If this applies to more than one provider, selecting the one with the smallest address achieves determinism. More advanced selection considerations require more information in the *sync* message. For example, if partners include a list of potential providers instead of their local best, the selection algorithm may calculate the distance between the proposed provider and all partners.

In case all *sync* messages are empty as none of the partners proposed a provider, the final selection result is empty. Each provider then searches and synchronises again. This time they know their partner addresses and do not require service layer routing.

IV. Experimental setup

The following simulation study explores synchronising parallel service flows and the impact of late binding on the failure probability and communication overhead of composites. We adopt a methodology similar to the one Atluri et al. [25] used and test seven requests types with seven services that vary in the number and length of AND-splitting

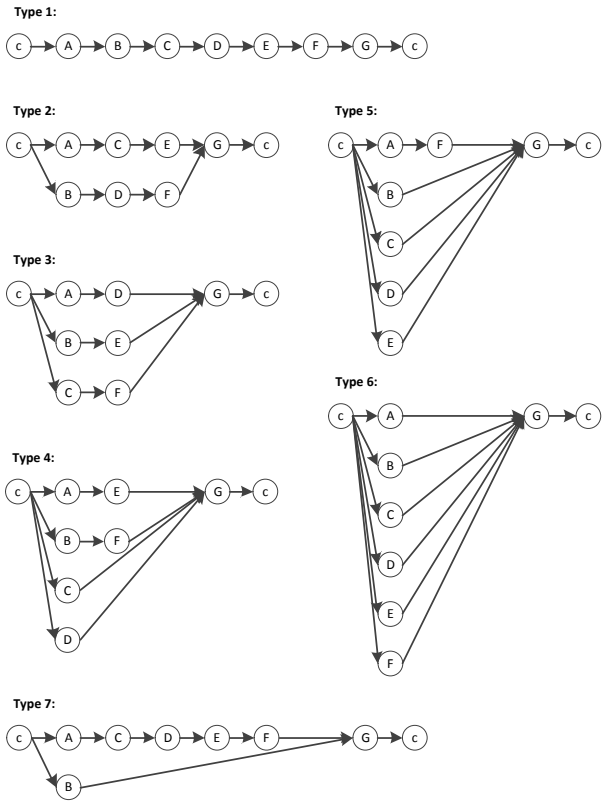


Figure 5. Request types with AND-splits and AND-joins

and AND-joining paths. Each request starts and finishes in the client c (Figure 5). Service providers represent a dense network of walking pedestrians in a city centre.

We compare early and late runtime binding and embed each in a composition model. CiAN, a workflow engine for mobile ad hoc networks [4], implements early binding in addition to beacon-based service advertisement, centralised fragmentation-based provider allocation, and decentralised service execution. The study uses CiAN*, our adaptation of CiAN, to relax the co-location requirement and apply AODV routing [26] rather than publish-subscribe data exchange. CiAN* delivers fragments using AODV routing because the original approach would fail if a service provider never moved into the client's range to collect its fragment. Further, despite our best effort to preserve the original publish-subscribe solution, we found that AODV routing incurred less communication overhead and used it instead to exchange service results between consecutive service providers. Late binding with service layer routing is implemented as part of an opportunistic composition model which features situational advertising and on-demand search, decentralised hop-by-hop selection, and decentralised interleaved execution. In this model directed broadcast, a combination of broadcast and unicast, allows for observing the composition process and update local knowledge about

Table I
SIMULATION CONFIGURATION

General	
Simulator	Jist/SWANS Ulm [27]
Simulation type	terminating
Field ($m \times m$)	500×500
Radio range (m)	100
Providers	149
Clients	1
Random	
Node placement, movement	Random Waypoint
Node speed (m/s)	1-2
Service execution time (ms)	10-100
Controlled	
Request type	1-7
Composition protocol	CiAN*, Opportunistic

service providers. Both composition models use the same non-standard description of a composite and select most recent neighbours first for required services.

In the simulation study (Table I) we recorded for a composition request whether it failed, the number of messages sent, and the response time. For each request type and composition model the simulation was repeated 100 times with different randomised initial settings. Each such setting was used in both models. We tested the most dynamic behaviour in which each required service must be bound to a unique provider.

V. RESULTS

The analysis of the simulation results examines whether late binding within the opportunistic composition model is less prone to failure, incurs lower communication, and makes better merge decisions than early binding in CiAN*.

A. Failure probability

A composition request has failed if it did not return the final composition result to the client. Overall, both composition models are equally likely to fail as their mean failure ratio over all request types is 5.5%. The opportunistic model is more affected by a particular request type than CiAN* (standard deviation 3.5% vs. 2%). The failure distribution graph in Figure 6 shows that opportunistic composition performs well for requests with primarily sequential parts (type 1,2,7) and struggles when the number of synchronisation partners increases (type 3,4,6). For CiAN* the variation among request types is less distinct as failure fluctuates between sequential request types (1 vs. 7) and also between parallel request types (5 vs. 6).

In this study, compositions fail due to stale routes or the collision of signals. CiAN* fails mostly because the route between the last service and the client has become invalid by the time the final result should be returned. Collisions occur rarely and are equally likely for all parallel request types. This suggests that network traffic is well-balanced. Opportunistic composition is prone to stale routes

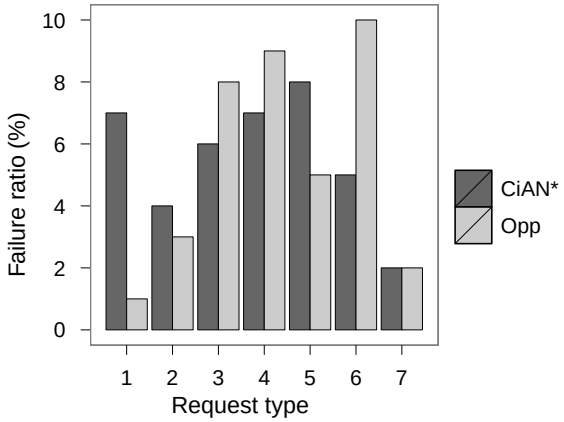


Figure 6. Failure distribution over request types

and signal collision. In particular, as parallelism increases, signals collide more often because multiple synchronisation partners deliver their results at the same time to the same merge node. Failure due to stale routes should be reduced by design as interleaved service execution establishes connections immediately before they are used. However, finding an agreement among multiple partners at runtime introduces delays during which already established routes may expire. The increase of stale routes and signal collisions lets the opportunistic approach fail more often than CiAN* when requests have a high degree of parallelism.

B. Communication overhead

The number of messages sent from the MAC layer until the composition completes successfully represents the communication overhead. Counting the number of sent messages on the MAC layer shows how many hops a message takes to get delivered and reflects the connectivity between the sender and receiver. Generally, the opportunistic model exchanges less messages than CiAN* (Figure 7a). The demand for communication in CiAN* is similar across all request types and highest during service discovery. In CiAN*'s proactive discovery strategy, a provider advertises its service offerings periodically and regardless whether a composition request was issued. The opportunistic model requires much less discovery messages because it searches for services only on request and potential providers sent advertisements only when they observed progress in the composition. The number of discovery messages decreases as the number of synchronisation partners increases because providers know the entire request and apply for multiple parallel services at once. As expected, the increase of synchronisation partners produces more synchronisation messages.

For requests with many synchronisation partners (type 3-6) the opportunistic model sends on average one token message less than CiAN* although both use the same selection algorithm. This indicates that multi-party selection may have a more comprehensive system view to select merge providers that are in close proximity to all partners.

We further evaluate service layer routing as a solution for late binding of a merge service. For this, comparison to CiAN* is not possible because it binds all services early and does not require agreement during execution. Instead, we compare our proposed routing solution against step-by-step path updating [3]. This late synchronisation strategy notifies parallel paths about binding decisions for each service that leads up to the merge service such that final synchronisation partners know of each other and can agree on a common provider. For such an updating strategy, the minimum number of messages to be exchanged is the number of parallel branches minus one multiplied by the total number of nodes between the split and merge node. This is based on the assumption that the sender and receiver are in direct communication range and that final synchronisation partners have to agree only once. Compared to such ideal conditions, service layer routing in the actual simulation performs well for type 2, 3 and 7 (see Figure 7b). The request types 4 to 6 motivate further simulation studies that test both approaches in the same dynamic ad hoc environment.

C. Response time

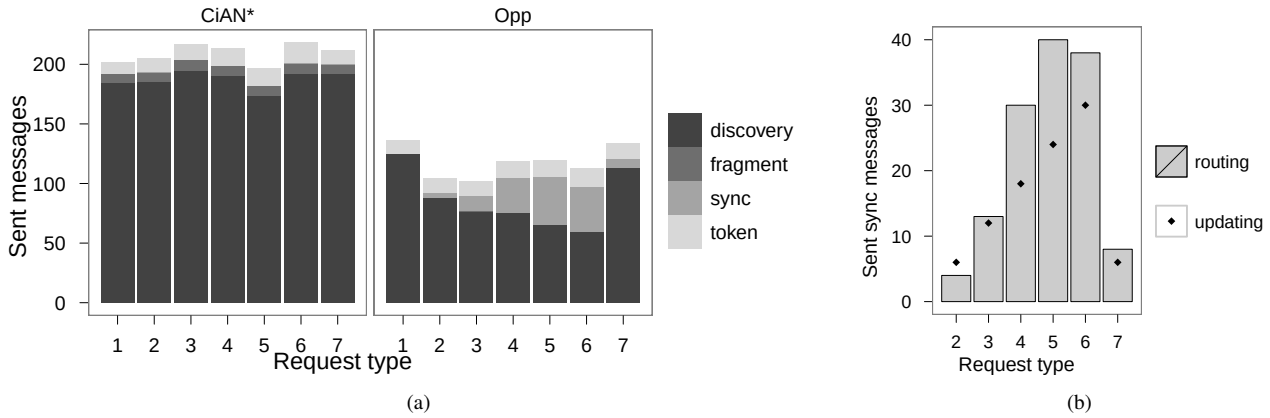
The response time is the delay on the client from sending the first fragment or composition request to receiving the final result. On average and across all request types CiAN* requires 4.9 seconds (standard deviation 3.5 seconds) and the opportunistic model 625 milliseconds (standard deviation 580 milliseconds) to complete a request (Figure 8).

The main reason for the big difference between CiAN* and opportunistic service composition is the time each model needs to establish a new route. The timeout for route requests is, in both models, two seconds plus one second per time-to-live which represents the search radius. If a route request fails, the search radius is incremented by two. The opportunistic model finds routing information within its one-hop neighbourhood before the first route request times out after three seconds. In CiAN* most cases timeout after three seconds without having found a route. The node then increases its search radius and with it the timeout.

The high standard deviation depicted as error bars reflect that many composition requests in CiAN* need to search their two and three-hop neighbourhood to establish a route. This suggests that some consecutive service providers in a request are not well-connected. Even if the route would be established from 1-hop neighbourhood information, the lower end of the error bars indicate that the opportunistic model would still finish more quickly than CiAN*.

VI. Discussion

Collaboration based on ad hoc networks as opposed to offloading operations into a stable platform is necessary if access to such platform is not available, incurs high delays



(a) (b)

Figure 7. Communication overhead (a) total and (b) for synchronisation

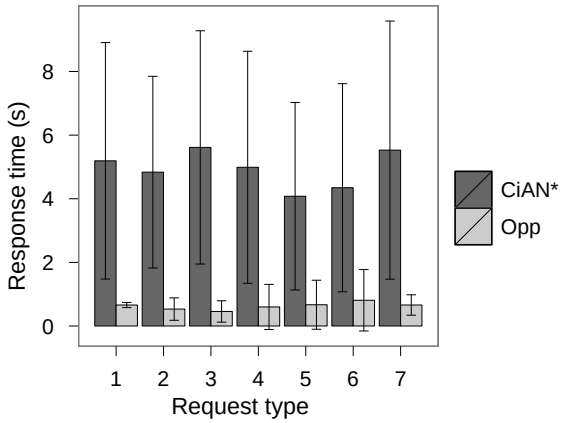


Figure 8. Distribution of response time over request types

and losses, or does not satisfy all requirements for real-time data. Service compositions in dynamic ad hoc environments need to remain flexible towards runtime changes and reduce communication to conserve scarce bandwidth and energy resources. The proposed multi-party late binding protocol addresses these needs and enables potentially unknown synchronisation partners in parallel service flows to agree on a common merge node. The simulation study for dense slow-moving networks shows that our approach is overall equally successful, less communication-intensive, and faster to respond than early service binding in a fragmentation-based approach. Passing on the binding control in a decentralised manner lets service providers select successors based on their connectivity which can potentially reduce time and communication effort. However, the protocol's success rate varies depending on the request structure. For requests with many parallel paths, the protocol becomes more prone to collisions and stale routes and performs worse than early binding. For moderate parallelism, on the other hand, late binding outperforms early binding.

The proposed protocol uses the structure and the executable part of the request to route synchronisation messages between unknown synchronisation partners. This way, a service provider discovers its partners in an ad hoc manner without prior search and informs all partners of its most suitable merge candidate without further communication. However, all providers between a split and merge service take on routing responsibility after they executed their allocated service. For them the risk of link failure due to mobility increases because they have to stay connected to their successors for a longer time. Similarly, these providers commit to a request for a longer time which limits their availability to other compositions. Future work will address this resource blocking problem and investigate how service routers can decide when to release bound resources.

Multi-party late service binding is part of an opportunistic model for service composition in dynamic ad hoc environments. The model explores ways to coordinate the discovery, allocation, and execution of complex service requests with the least amount of communication effort. Its design principle is to act on demand and defer any task that involves interaction with peers to the latest possible moment. In this context, we are currently testing our approach with further composition logic (e.g., XOR and OR) and for different load and mobility settings. We plan to evaluate the opportunistic composition model in different scenarios that vary in the density of service providers, speed of node movement, and the number of composition clients.

ACKNOWLEDGMENT

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

REFERENCES

[1] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140 – 150, 2010.

[2] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell, "Darwin phones: The evolution of sensing and inference on mobile phones," in *International Conference on Mobile systems, applications, and services (MobiSys)*. ACM, 2010, pp. 5–20.

[3] U. Yildiz and C. Godart, "Synchronization solutions for decentralized service orchestrations," in *International Conference on Internet and Web Applications and Services (ICIW)*. IEEE, 2007, pp. 39 –39.

[4] R. Sen, G.-C. Roman, and C. Gill, "CiAN: A workflow engine for manets," in *Coordination Models and Languages*, ser. LNCS, D. Lea and G. Zavattaro, Eds. Springer, 2008, vol. 5052, pp. 280–295.

[5] C. Schuler, R. Weber, H. Schuldt, and H.-J. Schek, "Scalable peer-to-peer process management - The OSIRIS approach," in *International Conference on Web Services (ICWS)*. IEEE, 2004, pp. 26 – 34.

[6] W. Yu, "Decentralized orchestration of BPEL processes with execution consistency," in *Joint International Conferences on Advances in Data and Web Management APWeb/WAIM*, ser. LNCS, vol. 5446. Springer, 2009, pp. 665–670.

[7] C. Groba and S. Clarke, "Opportunistic composition of sequentially-connected services in mobile computing environments," in *International Conference on Web Services (ICWS)*. IEEE, 2011, pp. 17–24.

[8] E. Philips, R. Van Der Straeten, and V. Jonckers, "NOW: A workflow language for orchestration in nomadic networks," in *Coordination Models and Languages*, ser. LNCS, D. Clarke and G. Agha, Eds. Springer, 2010, vol. 6116, pp. 31–45.

[9] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "Service composition for mobile environments," *Mobile Networks and Applications*, vol. 10, pp. 435–451, 2005.

[10] D. Martin, D. Wutke, and F. Leymann, "A novel approach to decentralized workflow enactment," in *International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2008, pp. 127–136.

[11] U. Yildiz, R. Badonnel, and C. Godart, "On service orchestration in mobile computing environments," in *International Conference on Services Computing (SCC)*, vol. 2. IEEE, 2008, pp. 545 –548.

[12] H. Fernández, T. Priol, and C. Tedeschi, "Decentralized approach for execution of composite web services using the chemical paradigm," in *International Conference on Web Services (ICWS)*. IEEE, 2010, pp. 139 –146.

[13] V. Prinz, F. Fuchs, P. Ruppel, C. Gerdes, and A. Southall, "Adaptive and fault-tolerant service composition in peer-to-peer systems," in *International Conference on Distributed Applications and Interoperable Systems (DAIS)*. Springer, 2008, pp. 30–43.

[14] Z. Wang, T. Xu, Z. Qian, and S. Lu, "A parameter-based scheme for service composition in pervasive computing environment," in *International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*. IEEE, 2009, pp. 543–548.

[15] X. Gu, K. Nahrstedt, and B. Yu, "Spidernet: An integrated peer-to-peer service composition framework," in *International Symposium on High performance Distributed Computing (HPDC)*. IEEE, 2004, pp. 110–119.

[16] E. Park and H. Shin, "Reconfigurable service composition and categorization for power-aware mobile computing," *Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1553–1564, 2008.

[17] F. A. Samimi and P. K. McKinley, "Dynamis: Dynamic overlay service composition for distributed stream processing," in *International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2008, pp. 881–886.

[18] S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic service composition in pervasive computing," *Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 907–918, 2007.

[19] M. Wang, B. Li, and Z. Li, "sFlow: Towards resource-efficient and agile service federation in service overlay networks," in *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2004, pp. 628–635.

[20] S.-H. Park, T.-G. Lee, H.-S. Seo, S.-J. Kwon, and J.-H. Han, "An election protocol in mobile ad hoc distributed systems," in *International Conference on Information Technology: New Generations (ITNG)*. IEEE, 2009, pp. 628 –633.

[21] A. Singh and S. Sharma, "Message efficient leader finding algorithm for mobile ad hoc networks," in *Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2011, pp. 1–6.

[22] S. Higashi, S. Ata, A. Nakao, and I. Oka, "Server selection for equalizing of performance in distributed cooperative system," in *International Conference on Information Networking (ICOIN)*. IEEE, 2011, pp. 519 –524.

[23] S. Roy, K. Herlugson, and A. Saberi, "A control-theoretic approach to distributed discrete-valued decision-making in networks of sensing agents," *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, pp. 945–957, 2006.

[24] A. Polyvyanyy, L. Garca-Bauelos, and M. Dumas, "Structuring acyclic process models," in *Business Process Management (BPM)*, ser. Lecture Notes in Computer Science, R. Hull, J. Mendling, and S. Tai, Eds. Springer Berlin / Heidelberg, 2010, vol. 6336, pp. 276–293.

[25] V. Atluri, S. A. Chun, R. Mukkamala, and P. Mazzoleni, "A decentralized execution model for inter-organizational workflows," *Distributed and Parallel Databases*, vol. 22, no. 1, pp. 55–83, 2007.

[26] C. E. Perkins, E. M. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (aodv) routing. [Online]. Available: http://www.ietf.org/rfc/rfc3561.txt

[27] JiST/SWANS Edition of Ulm University. [Online]. Available: http://vanet.info/jist-swans/download.html