A Distributed Constraint Satisfaction Problem Approach to Virtual Device Composition

Eric Karmouch, *Member*, *IEEE*, and Amiya Nayak, *Senior Member*, *IEEE*

Abstract—The dynamic composition of networked appliances, or virtual devices, enables users to generate complex, strong, and specific systems. Current MANET-based composition schemes use service discovery mechanisms that depend on periodic service advertising by controlled broadcast, resulting in the unnecessary depletion of node resources. The assumption that, once generated, a virtual device is to remain static is false; the device should gracefully degrade and upgrade along with the conditions in the user's environment, particularly the network's current performance requirements. Presently, schemes for infrastructure-less virtual device composition and management do not consider this adaptation. We present a distributed constraint satisfaction problem (distCSP) for virtual device composition in MANETs that addresses these issues together with simulations that show its effectiveness and efficiency.

Index Terms—Pervasive computing, multimedia applications, algorithm/protocol design and analysis, mobile environments, constraint satisfaction

1 Introduction

Continuous network connectivity and the increasing availability of online data are creating a demand from consumers for networked services to be seamlessly integrated into their lifestyles. This is driving network and computing technology into everyday objects, and bringing to fruition the concept of the *networked appliance* [1]. A networked appliance is defined as a dedicated function consumer device with an embedded processor, a network connection, and the ability to disperse its capabilities within the network. This allows other devices to combine with and use those capabilities as part of a *virtual device*, a collection of heterogeneous devices in the vicinity of the user that behave as a single homogeneous device in solving a given task. The functionality of a networked appliance is distributed, enabling it to be controlled, monitored, managed, and extended beyond what it was initially designed to do. More specifically, as this paper discusses, a virtual device is defined as a system of strong networked devices, both fixed and mobile, with many distributed input and output capabilities and able to provide users with coherent and surrounding interfaces. The devices are specific, meaning that they provide a single functionality at high quality. Users are able to incorporate autonomous devices in their vicinity as needed. Single applications can be distributed across a number of devices, with the strengths of each device providing enhanced user experience and quality of service (QoS). Virtual devices are managed automatically, with little or no human involvement. For more details regarding virtual devices and challenges they pose, see Supplemental Section 1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.39.

An open problem is how to achieve the automatic composition of virtual devices and the services they provide in order to create dynamic service configurations with limited human intervention. The aim is to automate the discovery process and to enable the devices to determine what services are required. Various approaches have been taken in the home networking research area [2]. They range from advanced human-computer interaction to networking and automatic interoperation. However, these approaches tend to be orthogonal to the development of enabling platforms. They manage only to provide basic interoperability through centralized approaches, resulting in little or no support for mobile ad hoc network environments (MANETs). The hope of finding more reactive and decentralized solutions has lead to research that specifically targets service composition in MANETs [3], [4], [5]. Little or no attention has been paid to quality of service. As a result, finding the "best composition possible," as opposed to "a composition that fits," is not considered a priority. This research also relies on discovery mechanisms that are largely based on constant periodic service advertising by controlled broadcast, whether composition is currently desired or not. The effect is that node resources such as battery and processing power are unnecessarily depleted. Moreover, an ad hoc network is composed of a collection of wireless nodes, some or all of which may be mobile; a wireless network is dynamically created without aid from either infrastructure or administration. Ad hoc networks can therefore be described as self-creating, self-organizing, and self-administering, offering unique benefits and versatility to the virtual device problem. However, they also create environments in which measurable attributes such as bandwidth, delay, loss, and jitter are highly likely to vary over time. A composed virtual device should therefore gracefully degrade and upgrade

• *The authors are with the School of Electrical Engineering and Computer Science at the University of Ottawa, ON K1N 6N5, Canada. E-mail: ekarm041@eecs.uottawa.ca, anayak@eecs.uottawa.ca.*

Manuscript received 6 Dec. 2010; revised 8 July 2011; accepted 22 Jan. 2012; published online 27 Jan. 2012.
Recommended for acceptance by J. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-12-0710.
Digital Object Identifier no. 10.1109/TPDS.2012.39.

along with the performance of the network and with minimal interruption to the user.

In this paper, we define the virtual device composition problem and present the Virtual Device Constraint Satisfaction Protocol (VDCSP). This does not require broadcast-based service advertisements. It also includes a graceful degradation and upgradation that corresponds to the performance of the network. We achieve this by modeling and solving the virtual device problem as a *distributed constraint satisfaction problem (distCSP)* [6]. A CSP is defined by the triplet *(X, D, C)*, where $X = \{X_1, \ldots, X_n\}$ is a set of n variables, $D = \{D_1, \ldots, D_n\}$ is a set of n domains of values, where D_i is the domain of the values of the variable X_i, and $C = \{C_1, \ldots, C_j\}$ is a set of j constraints of the problem. A CSP is solved by assigning values to the variables so that all constraints are satisfied. A distributed CSP is defined as a CSP where variables and constraints are distributed among automated agents. Solving a CSP therefore implies achieving coherence or consistency among agents. Another benefit of the distCSP approach is that, unlike traditional schemes, distCSPs can be solved without the need for agents to directly divulge complete or precise information about their domain and constraints. Instead, information is passed in a highly summarized form. This is important for privacy and security reasons, as there may be information pertinent to composition, such as the client billing model, that a node may not wish to divulge.

Our preliminary work on the possibility of using CSPs for virtual device composition is presented in [7],]8]. The contributions of this paper are 1) to modify our solution to include network attributes in the process, 2) to extend our solution to account for postcomposition adaptation based on changing network attributes, and 3) to provide more extensive simulations that study the effects of mobility and scalability on the efficiency and effectiveness of the protocol. The simulations show that VDCSP is able to compose high-quality virtual devices that meet task requirements, service constraints, and user preferences, and that also perform postcomposition adaptation, while minimizing cost in messages used and composition time.

The rest of the paper is organized as follows: Section 2 introduces related work on both infrastructure-based and infrastructure-less environments. It includes a detailed description of a prominent distributed service composition protocol (DSC) that we use for comparison in simulation. Section 3 presents our VDCSP. We incorporate network attributes into the service selection and composition process, and extend the protocol to account for postcomposition adaptation based on changing network attributes. Section 4 provides the findings of our simulations, showing the effectiveness of our method and its response to changing mobility and scaling.

2 Related Work

Service discovery and composition is a significant area of research [9], [10], [11], [12], [13], [14], [15], [16], [17] in the context of web services. Research in this area takes two paths: 1) service description and matching and 2) service discovery and composition architectures, mostly in infrastructure-based environments. Research into infrastructure-less environments is still relatively new. We begin by examining service discovery and composition in both kinds of environments. We then provide a more detailed description of Chakraborty et al.'s Distributed Service Composition Protocol [3], the protocol we use for comparison.

2.1 Infrastructure-Based Environments

Languages for describing web services in a dynamic manner include the Web Services Development Language (WSDL) [18] and the DARPA Agent Markup Language for Services (DAML-S) [19]. WSDL is an XML-based language in which network services are described as a collection of endpoints operated by document/procedure-oriented messages. DAML-S focuses on the standardization of the Web Ontology Language (OWL) [20]. OWL can be defined as a set of XML elements and attributes, which, through standardized meanings, are used to define terms and their relationships. Languages for formally specifying services and composite services include the XML-based Web Services Flow Language (WSFL) [21] and the Business Process Execution Language for Web Services (BPEL4WS) [20], used in the formal specification of business processes and business interaction protocols. Complex planning engines have also been developed using service descriptions to generate declarative specifications of workflows for the composition of services [23], [25].

The majority of service discovery and composition architectures are designed for wired infrastructures, using central lookup servers for service registration, discovery, and composition, and assuming stable nodes connected by reliable communication channels. These include Jini [25], Salutation and Salutation-lite [26], UPnP [27], and the Service Location Protocol [26], and others described in [29], [30]. These techniques often involve preconfigured composition managers on dedicated machines that require a large amount of memory, bandwidth, and processing power. They do not cater to highly pervasive, mobile and ad hoc environments, leaving themselves vulnerable to issues such as central points of failure, mobility, and fault management.

2.2 Infrastructure-Less Environments

As previously mentioned, service discovery and composition in infrastructure-less environments is a relatively new area of research. Existing work [31], [32] relies too heavily on broadcast-based techniques that lead to scalability and efficiency issues. The work also tends not to include QoS metrics for enhanced composition. As a result, the main focus of most existing protocols is to discover, integrate, and execute services while satisfying resource constraint issues typical of MANETs (such as connectivity, routing, and energy awareness) while failing to address the quality of the composed service. However, the work of Chakraborty et al. [3] makes significant contributions in the area. Their group-based service discovery protocol (GSD) is based on peer-to-peer caching of service advertisements and on group-based selective forwarding of service discovery paths. The discovery and integration allows for composite services, but the approach does not deal with specific user and task needs and situations.

More recent techniques include the work of Han and Zhang [33] and Wang [34]. Han et al. design and implement a service composition protocol based on the Dynamic Source Routing (DSR) protocol. It goes beyond best-effort approaches and considers QoS in real-time systems. Although their solution considers traditional QoS parameters such as delay and cost, they do not consider user-based qualities of a virtual device. Wang provides a solution that incorporates a service providers' mobility prediction. This provides for more reliable composition in that, in terms of mobility, the most stable service can be selected from several services of the same type. This is an important aspect of composition that provides a valid extension to the work presented in this paper. But it does not consider the virtual device problem as a whole.

Standardized protocols, such as the Bluetooth Service Discovery protocol [35], when extended into ad hoc environments, also rely heavily on broadcast, and support only basic identifier matching. More recent work is promising, including the MobiLife Integrated Project [36], Intel Research's Dynamic Composable Computing [37] project, and Namman et al.'s Flexible Service Composition Framework [38]. All three address decentralized ad hoc compositions tailored to specific user needs and situations.

The MobiLife Integrated Project brought advances in mobile applications and services within reach of the everyday lives of users by developing new applications and services based on the evolving capabilities of 3G systems and beyond [36]. The Project solved parts of the virtual device problem with its contributions in the area of multimodality and personalization. Theoretical frameworks were developed to satisfy the mobile user experience. A strong emphasis was placed on making the best use of available environment devices. MobiLife sets itself apart from similar projects by including mobile devices as core components and by its concern for device mobility and the need for continuous adaptation. However, much of this work remains theoretical.

Dynamic Composable Computing (DCC), introduced by Intel Research, explores the impromptu assembly of a logical computer from the best set of wireless parts available nearby [37]. The work builds on existing standards to develop an architecture that enables users to quickly connect mobile devices. In particular, Intel Research develops a layer-2 service discovery scheme [37] that integrates IP service advertisement and the discovery process. This reduces the inefficiency of broadcast discovery techniques. The approach still relies on manual/named compositions that require heavy user involvement as compared to [35], [36], 38], although future work will investigate ranking-based compositions.

Namman et al.'s Flexible Service Composition Framework [38] uses a hybrid (centralized/distributed) approach. As default, a centralized technique enables automatic service compositions in heterogeneous environments. Should the central server become unavailable, local information is used for the composition process in combination with an interrogation procedure between neighboring nodes. Although this provides adequate support in an environment with limited interruption at the centralized server, the technique would inevitably suffer in an environment too dynamic for a centralized server to exist at all.

The use of a Constraint Satisfaction Problem (CSP) in service composition, providing automated candidate service selection, appears in some research [39], [40]. This paper also uses a distributed Constraint Satisfaction Problem within a MANET.

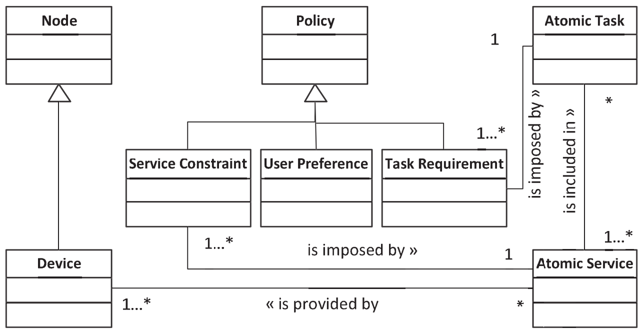
2.3 Distributed Service Composition Protocol

As previously described, Chakraborty et al.'s distributed service composition protocol [3] is based on the principles of reactive systems. It consists of mobile nodes, representing various devices in the environment that provide one or more services and are able to be invoked by peer nodes. These nodes are connected using ad hoc network protocols; the discovery and integration of services allows for composite services. Although nodes may differ in physical properties such as computational power and battery life, there is no differentiation between a requesting source, service provider, or broker. A requesting source may simultaneously act as a service provider and/or a broker for separate composition requests. The process of composition begins by a requesting source electing a broker within the network. The broker then discovers nodes providing the required services (service providers), and integrates and executes those services on behalf of the source.

The protocol has four phases. The first phase is *broker arbitration* where the requesting source elects a broker, maximizing a potential value and considering each potential broker's resources and the service providers in their vicinity. This information is calculated by each potential broker and sent to the source. The elected broker is ideally a node with high computational power, low computational load and long battery life, and located in an area of high service density. Information on service providers in a potential broker's vicinity is obtained as each broker examines its cache, which is updated according to the group-based service discovery protocol [41]. GSD is based on two concepts: 1) peer-to-peer caching of service advertisements within a limited vicinity and 2) group-based selective forwarding of discovery requests. Each service provider periodically advertises a list of their services to other peer nodes within a limited vicinity, including a list of service groups. The process involves the hierarchical grouping of services; grouping information is used to selectively forward discovery requests to service providers. The effect is that network-wide broadcasts are reduced and efficiency in discovering services is increased.

The second phase is *service discovery.* Using GSD, all necessary services are discovered, based on the matching of requests with cached descriptions of services. Services are represented using DAML-based semantics [19]. This kind of matching can identify two services of the same type, but it is limited in identifying the variation between those two services, the current availability of each, and the importance of these factors to the user.

The third and fourth phases are *service integration* and *service execution.* In these phases, discovered services are selected. No specific algorithm is used; selection is based on a number of basic cost factors, such as "nearest available service." An execution flow is created identifying relevant service information such as node binding, control flow, and network parameters. Execution occurs in a distributed manner: the broker executes service and transmits information from the previous service to the next. This star-shaped



(a)

TCspec Task: Video Conferencing
<u>Task Requirements</u>
-P1- Video input service S1 is required
-P2- Audio input service S2 is required
-P3- Video output service S3 is required
-P4- Audio output service S4 is required
<u>Service Constraints</u>
-P5- Services S1-S4 must reside in close proximity
<u>User Preferences</u>
-P6- Service S1 resolution must be ≥ 700x500
-P7- Service S3 resolution must be ≥ 1000x800
-P8- Wait time to utilize a service must be ≤ 1min

(b)

Fig. 1. (a) Environment class diagram. (b) Sample TCspec for a video conferencing task.

execution continues according to the requirements of the execution flow.

3 Virtual Device Composition

In Fig. 1a, we show an environment made up of mobile nodes, connected by ad hoc network protocols, that represent various devices providing one or more services and that can be invoked by peer nodes. Device composition involves the composition of services offered in a user's vicinity and provides the user with a virtual device, satisfying what we describe as an atomic task. An *atomic task* is an application-level task, independent from any other and requiring one or more atomic services. An *atomic service* is a service residing on a single node, with further components only if they too reside on the same node. Generated from a task is a *task-based composition specification (TCspec)*, a specification of service and service property requirements represented as policies (Fig. 1b). Policies are divided into three categories: task requirement, service constraint, and user preference. *Task requirements* represent the minimum services required for task satisfaction. *Service constraints* represent the constraints needed to allow the composed service to function as intended. *User preferences* are service constraints controlled by the user; they simply shape the desired QoS, meaning that, at any instant, a number of service composition permutations in the environment satisfy a TCspec. The QoS associated with one particular composition separates it from other compositions. In this paper, we define QoS as the ability of a virtual device to match the predetermined task requirements, service constraints, and user preferences (the TCspec). Our objective is to efficiently examine possible composition configurations in a user's vicinity and to identify the configuration that 1) satisfies all service requirements, 2) adheres to all service constraints, and 3) best caterers to the user's preferences. Complicating this process is the fact that nodes are independent and may be operating under different rules and guidelines in making composition decisions. We refer to these rules and guidelines as *private policy,* (information used by a node), in addition to *public policy* (information in a TCspec), to decide how to react in satisfying tasks. Private policy may refer to security rules, business models, priority guidelines, and so on. Private policy information is confidential and should not be shared with other nodes. This makes centralized solutions that involve extensive information-sharing unsuitable. An acceptable solution is distributed, providing efficient negotiation techniques while sharing a minimal amount of information.

As previously mentioned, ad hoc networks are known for their highly dynamic nature. It is incorrect to assume that a composed virtual device will remain valid in a static form. As the network changes, the virtual device should adapt, but in a way that provides minimal disruption to the user. In the following discussion, we use the performance metrics of bandwidth, delay, loss, and jitter to develop a QoS model that enables the virtual device to adapt to the variations of the network.

3.1 Virtual Device Composition DistCSP

In order to meet the objectives outlined in the previous section, we model virtual device composition in the form of a distributed constraint satisfaction problem $VDC\text{-}DistCSP = (X, D, C)$ where:

- $X = \{X_i\}(i = 1, \dots, n;\ X_i \subseteq D)$. X_i is a variable corresponding to the set of services being provided by node i to satisfy a particular task.
- $D = \{D_i\}(i = \{1, \dots, n\})$. D is a set of n domains of values, such that D_i is the service option domain of X_i (i.e., the set of services that X_i can provide).
- C is the set of constraints of the problem. We can formulate them as follows:

$$\forall_{i,j} \quad X_i \cap X_j = \{\}\ i, \quad j \in \{1, \dots n\}, \quad i \neq j, \tag{1}$$

$$\forall_i \cup X_i = \{S\}, \tag{2}$$

$$PR(\{S\}, B, L, D, J) \geq \lambda, \tag{3}$$

$$\{S\} = Max(QoSscc), \tag{4}$$

where S is a set of services equivalent to all k required services d_j, $j = \{1, \dots, k\}$, where each service corresponds to a particular task requirement policy of a TCspec. Constraint (1) guarantees that at most one node provides any particular service. Constraint (2) ensures that all required services have been assigned. Constraint (3) ensures that the performance requirements *PR* of the composed service (bandwidth (BW), delay (D), loss (L), and jitter (J)) are more constrained than or equally constrained as the current performance requirement λ of the network. Constraint (4) ensures that the chosen service set S maximizes $\mathrm{QoS_{vd}}[\lambda]$, the overall QoS of the composed virtual device.

Similar to Chakraborty et al.'s solution, we use a broker (as defined in Section 2.3). By electing a broker (B), we denote a utility function $U(B_i)$ as the utility value of each potential broker as:

$$U(B_i) = W_L \cdot L(B_i) + W_{CR} \cdot CR(B_i) + W_P \cdot P(B_i), \quad (5)$$

where $L(B_i)$ is the battery life of B_i, $CR(B_i)$ is the number of composite requests currently being processed by B_i, and $P(B_i)$ is the processing power of B_i. Applied to each parameter is a weight identifying the amount of influence given to that parameter during the computation of $U(B_i)$. We are aware of the difficulties that may arise when determining values for the weights. We therefore consider this issue out of the scope of the paper. Machine learning techniques may help in finding reasonable values for some of the weights.

In identifying the QoS of a particular composition, we apply a model similar to that defined by Perttunen et al. [42]. However, we take a decentralized approach that enables the model to function in MANETs. Requirements of the virtual device concept are 1) being aware of the user's continuously changing environment, 2) immediately recognizing the user's expectations, and 3) providing a service that matches the user's circumstances. We structure a QoS function in the context of virtual device composition and identify it as the degree of match between the requirements of a user's task (including service constraints, satisfying both task requirements and user preferences) and the qualities and capabilities of service composition. We also identify the necessary spectrum of tolerance in bandwidth (BW), delay (D), loss (L), and jitter (J) that a particular composition can accept. Given this information, potential compositions, referred to as service composition candidates (SCCs), can be further classified by their performance requirements. We express this function as $QoS_{vd}[PR]$, representing the maximum QoS among all service composition candidates. This satisfies the performance requirement PR where

$$QoS_{vd}[PR] = Max(QoS_{scc})[PR], \quad (6)$$

$$PR_{scc} = PR(\forall S_i \in SCC, Max(BW), Min(L), Min(D), Min(J)), \quad (7)$$

$$QoS_{scc} = A_w \cdot \sum_j W_{tj,u} \cdot Sim_{tj}(d_j, d_{j,tcspec}), \quad (8)$$

$$A_w = Min(A_{ws}), \text{ and} \quad (9)$$

$$A_{ws} = \det_{avail}(ST_s, P_s, Q_s, \ldots). \quad (10)$$

Using a similarity function *Sim*, we are able to classify similarities in quality and capability between services in a TCspec and those available in the user's environment. Moreover, Sim_{Ti} is a precise similarity function for a specific service or service property d_j of type T_j. The output of the similarity function is a normalized value between [0,1], considering both positive and negative discrepancies from a precise match. The nearer the value is to 1, the higher the degree of match. In shaping the amount of relevance given to a particular service or service property type, each iteration of the summation is affected by a user-specific or default weight $W_{ti,u}$, representing the quantity of weight given. A second weight A_w is applied, representing the availability of the potential service composition. A_w is attained by computing the service availability A_{ws} for each service in the potential composition, and computing the minimum (9). The calculation of A_{ws} is founded on various factors including the current state of the service ST_s, the usage policy of the service P_s, queue information Q_s on other users waiting for the service, and other relevant information (10). Every service S_i has a performance requirement formed as a bound PR_{Si} representing the minimum bandwidth and maximum loss, delay, and jitter it is able to tolerate (8). The performance requirement of a particular *SCC* can be computed from the maximum *BW* and minimum L, D, and J values across all services involved (7).

We are aware that defining similarity functions for each service and service property type is an exhaustive approach; potentially hundreds of types may exist. In real-life applications, determining values for weights and selecting the most appropriate similarity functions is a complex issue. Considering the ontological relations of the service types could help with the calculation, especially if generic ontological relations could be used. But we consider these issues out of the scope of this paper: solutions may be found in machine learning techniques.

To further clarify, this sample similarity function is used in identifying optimum resolution, in that both larger and smaller resolutions are handled as worse than an exact match with a service:

$$Sim_{res}(R_1, R_2) = \left(\frac{Min(x_1, x_2)}{Max(x_1, x_2)} + \frac{Min(y_1, y_2)}{Max(y_1, y_2)}\right) \Big/ 2, \quad (11)$$

where R_1 and R_2 are the values of the resolution properties, and x_1, x_2, y_1, and y_2 are the respective horizontal and vertical resolutions. The division by two normalizes the similarity to [0,1].

Having identified our distCSP, we apply an algorithm based on the asynchronous backtracking algorithm [6]. We call our method the VDCSP and our algorithm the *Virtual Device Constraint Satisfaction Algorithm* (VDCSA).

3.2 Virtual Device Constraint Satisfaction Protocol

The VDCSP allows automated agents, each representing the interests of a particular node, to act concurrently and asynchronously. No global control is required in deciding how to best form the virtual device that the user needs. The protocol requires no periodic service advertising and is able to incorporate private policy.

The virtual device composition problem, from a distCSP perspective, can be represented as an $m \times n$ Matrix A (Fig. 2), comprised of m nodes and n services:

- A black circle in $A_{x,y}$ represents a service $d = S_y$, such that N_x provides service d.

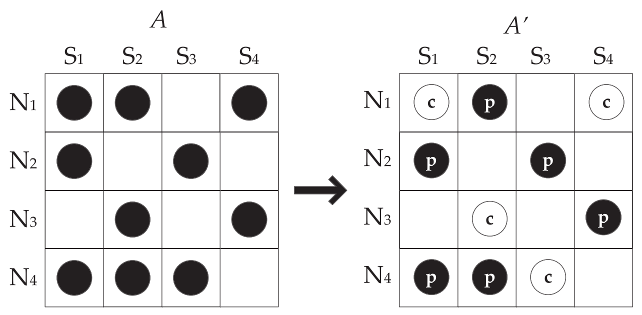


Fig. 2. A transformation from Matrix A to $\mathrm{A'}$, visually abstracting the virtual device problem. ($\mathrm{c = committed}$, $\mathrm{p = potential}$).

The solution requires the application of the constraint set C defined in Section 3.1, whereby a Matrix $A'_{x,y}$ (Fig. 2) is formed:

- A black circle in $A_{x,y}$ containing the letter p represents a *potential* service $d = S_y$ in a node N'_xs X_x variable, such that N_x has the potential of contributing service d in the virtual device composition.
- A black circle in $A_{x,y}$ containing the letter c represents a *committed* service $d = S_y$ in a node N'_xs X_x variable, such that N_x has committed to contributing service d in the composition. The attribute c is used as a composition-specific flag to indicate which nodes are currently committing a service. Should a node's resources allow it to do so, it may commit to a number of compositions (and virtual devices) simultaneously. The commitment is dynamic and may change throughout the lifetime of the virtual device.
- Each column n must contain *exactly one* black circle containing the letter c.

The composition of node/service pairs corresponding to the black circle values containing the letter c form a composition satisfying the constraint set C. VDCSP establishes a Matrix A and provides a concurrent and asynchronous negotiation mechanism to transform Matrix A into Matrix $\mathrm{A'}$. The following shows the various steps of VDCSP.

3.2.1 Candidate Formation

Candidate formation is the first step of the protocol and allows for the creation of Matrix A. When a task becomes activated on a user's device, the corresponding node sends a *virtual device request (VDrequest)* message into the user's immediate vicinity, using a bounded broadcast (Fig. 3). A VDrequest contains the task's TCspec, which is used by a receiving node to decide whether or not it can contribute a service to the task requirements. A node that can do so can, by implication, also meet the current performance requirement λ of the network. These nodes send a *virtual device reply (VDreply)* message to the VDrequest initiator. The VDrequest initiator uses these replies to compute a *candidate table* containing the addresses and unique identifiers of all responding nodes. The nodes in the candidate table are organized in *descending priority order* based on their unique identifiers. Once the table is formed, the VDrequest initiator distributes it to all candidates.

Fig. 3 shows an example of candidate formation. A task has become activated at Node N_0, at which time it

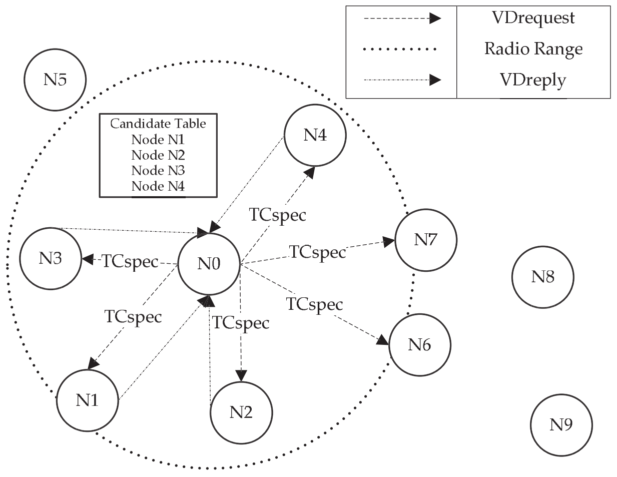


Fig. 3. A sample candidate formation scenario.

broadcasts, with bounded value of one hop, a VDrequest containing the corresponding TCspec. Nodes N_1, N_2, N_3, and N_4 determine that they are candidates and can provide one or more services listed in the TCspec. They reply with a VDreply message. N_0 then forms a prioritized candidate table and distributes it to all candidates.

3.2.2 QoS Ranking

By implication, a node receiving a candidate table is participating in the VDCSP. For each service d_j that the node N_i has agreed to potentially contribute, a *QoS ranking* is computed using (8) as $Rank_{qos} = A_{ws} \cdot W_{ti,u} \cdot Sim_{ti}(d_j, d_{j,tcspec})$, where a similarity function is computed between d_j and the service requirements specified in TCspec. The availability and user-specific/default weights A_{ws} and $W_{ti,u}$, of the service d_j are applied to the similarity function.

For each service a node has agreed to potentially contribute, three pieces of information are kept in a tertiary: service type, QoS ranking, and commitment status. The *commitment status (ComStatus)* is a variable, either *potential* (p) or *committed* (c), that represents whether or not the service has been committed by a node. The p and c values are shown in Fig. 2. For each of a node N'_is services, a tertiary is stored in N'_is X_i, such that $X_i = \{(d_j, Rank_{qosj}, ComStatus_j)\}(j = \{1, \ldots, k\})$.

Fig. 5a represents a global view of the Matrix A formed as described in the previous section, including the corresponding QoS rankings. For node N_1, $X_1 = \{(S_1, 100j, c), (S_2, 95, c), (S_4, 100, c)\}$.

3.2.3 Virtual Device Constraint Satisfaction Algorithm

With Matrix A complete, we apply the VDCSA. This is based on the traditional asynchronous backtracking algorithm for solving distCSPs [6]. The execution of VDCSA (pseudocode provided in Fig. 4) is asynchronous between candidate nodes; it begins with all candidates committing to all services in their respective X'_is and exchanging these values in *ok?* messages.

Ok? messages are used to spread knowledge of value assignments to candidate nodes, which use them to form their *agent view*. An agent view reflects a node's current view of the *partial solution*. A partial solution is a subset of the *final*

```
when received (ok?, [$N_i$,$X_i$]) do – (i)
   add[$N_i$,$X_i$] to agent_view;
   check_agent_view;
end do;

when received (nogood, $N_i$, nogood) do – (ii)
   add nogood to nogood_list;
   check_agent_view;
end do;

procedure check_agent_view – (iii)
   when agent_view and $X_{self}$ are not consistent do
      if no value in $D_{self}$ is consistent with agent_view then
      backtrack;
      else
         select $d \in D_{self}$ where agent_view and $d$ are consistent;
         $X_{self} \leftarrow d$;
         send (ok?,($N_{self}$, $X_{self}$) to candidates;
      end if;
   end do;

procedure backtrack – (iv)
   nogoods $\leftarrow$ {V | V = inconsistent subset of agent_view};
   when an empty set is an element of nogoods do
      broadcast to all agents that there is no solution, terminate this algorithm;
   end do;
   for each V $\in$ nogoods do
         send (nogood, $N_{self}$, V) to $N_j \in$ V
   end do;
```

Fig. 4. Partial VDCSA pseudocode.

solution, which is expanded by adding committed variable tertiaries one by one, until a final solution is reached. Different nodes may have different agent views at any given time, although as the algorithm progresses, all nodes work toward a single common agent view (the final solution).

Following the receipt of an *ok?* message (Fig. 4i), a node N_i compares its X_i with its agent view (Fig. 4iii), looking for any inconsistencies with regards to *higher priority nodes*. Inconsistencies occur when N_i's X_i value violates the set of constraints C of the problem. If inconsistencies are present, N_i attempts to resolve them by altering its X_i. If N_i is able to achieve consistency, it distributes its new X_i in an *ok?* message to all candidates. If no X_i exists, then it is necessary to *backtrack* (Fig. 4iv).

Backtracking involves sending one or more *nogood* messages to the *higher priority nodes* causing the inconsistency. A *nogood* message specifies exactly which tertiary value in X_i to consider as *nogood*. Every candidate keeps a *nogood list*, which adds to its constraint set C. On receiving a *nogood* message, the recipient attempts to adapt its respective X_i value, enabling the sender to reenter a consistent state (Fig. 4ii). If the *nogood* recipient is unable to resolve the inconsistency, then it too triggers a backtrack. In this way, *nogood* messages flow up the candidate table until they are either resolved or a no solution decision is reached. A no solution occurs when the highest priority node cannot resolve a *nogood*, such as when it has exhausted

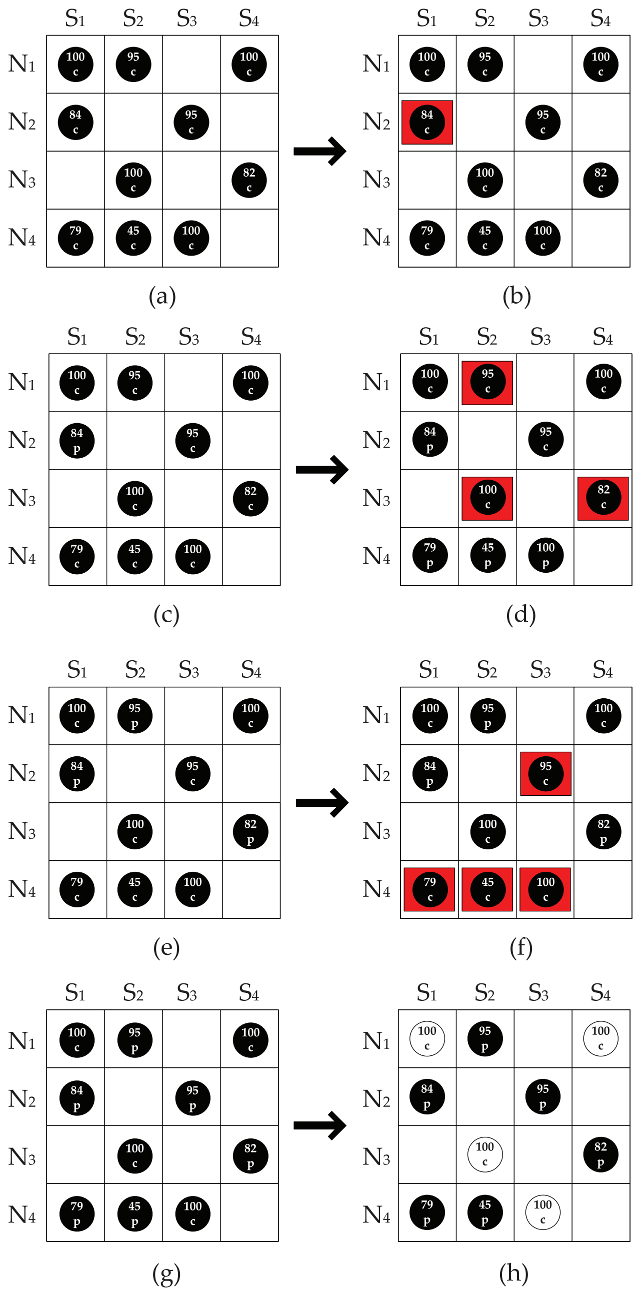


Fig. 5. Example execution of VDCSA.

all of its variable options. The problem is then overconstrained and the algorithm is terminated.

A solution occurs when all nodes share a common agent view and are in a consistent state. Nodes continue to periodically send *ok?* messages until either a no solution or final solution is reached.

3.2.4 VDCSA Example Execution

To clarify the algorithm, we provide an example. The trace of the algorithm's execution can vary according to the timing/delay of messages. This example shows one possible trace.

The initial values are shown in Fig. 5a. Priority has been determined by numerical identifiers, QoS rankings have been computed, all candidates have committed to their services, and these values have been exchanged using *ok?* messages. Each candidate now attempts to reach consistency.

Being the highest priority node, N_1 is currently in a consistent state, but N_2 is not. N_2 observes that N_1 has committed to service S_1 (Fig. 5b). In order to enter a consistent state, N_2 must uncommit to S_1 because N_2 does not provide a higher QoS ranking than N_1. However, N_2 keeps its commitment to S_3, and N_1 does not provide service S_3 (Fig. 5c).

N_3 is also in an inconsistent state (Fig. 5d). N_3 is able to rectify its S_4 value by uncommitting; however, since its QoS ranking for S_2 is higher than that of N_1, it is unable to uncommit to S_2. Therefore, N_3 sends a *nogood* message to N_1. N_1 receives the *nogood* message, and uncommits to S_2, allowing N_3 to enter a consistent state (Fig. 5e).

N_4's values for S_1, S_2, and S_4 are inconsistent. N_4 uncommits to services S_1 and S_2, but is unable to uncommit to S_3 (Fig. 5f). It sends a *nogood* to N_2; this allows N_4 to reach consistency (Fig. 5g) and a solution is found (Fig. 5h).

3.2.5 VDCSA Soundness and Completeness

The soundness of VDCSA is guaranteed because agents can only reach a stable state only if all of their variable values satisfy all constraints. The algorithm is also complete, finding a solution if one exists and terminating if no solution is found within a finite time.

A "no solution" scenario occurs if the problem is overconstrained. VDCSA responds to overconstrained problems by eventually generating a *nogood* depicting an empty set, in which any set of variable values leads to a constraint contradiction. This is a failure and causes the algorithm to terminate. The only way in which termination would not happen in finite time is if one or more nodes were to cycle through their possible variable values in an infinite loop. By induction, we show that this cannot happen.

If the node in an infinite loop is of highest priority, it will not send *nogood* messages, but will only receive them. When the node receives a *nogood*, it will change its value so that it either receives, or does not receive, a *nogood* in return. Should the node receive a *nogood* for all of its potential values, an empty set *nogood* will be generated. Should the node not receive a *nogood* after changing its variable value, no further changes will occur. Either way, an infinite loop cannot happen.

Now we assume that nodes X_1 to $X_{k-1} (k > 2)$ are in a stable state, and that node X_k is in an infinite loop. Since X_1 to X_{k-1} are stable and X_k only receives *nogood* messages from lower priority nodes, all *nogood* messages received should be resolved by a change in X_k's variable value. Because X_k's potential variable values are finite, two situations can occur: either X_k changes its value and does not receive a *nogood* in return or it exhausts its potential values and sends a *nogood* to $X_{1\ldots k-1}$. The first situation presents a contradiction in that X_k would not be in an infinite loop, and the second situation presents a contradiction in that $X_{1\ldots k-1}$ cannot receive a *nogood* if they are in a stable state. Therefore, X_k cannot be in an infinite loop and must terminate in finite time.

With regards to complexity, constraint satisfaction is generally considered NP-complete [6]. As a result, worst case time complexity is exponential in the number of variables n, $X = \{X_i\} (i = 1, \ldots, n; X_i \subseteq D)$.

3.2.6 Postcomposition Adaptation

As previously mentioned, should network conditions no longer satisfy the service composition initially selected, adaptation is needed. In this case, as the broker periodically monitors network conditions, should a λ occur that the composed virtual device cannot satisfy, or should λ allow for a service upgrade, a new virtual device request is sent forming a new candidate table and triggering another iteration of VDCSA.

Applying the preceding QoS model assumes that an advertised service contains performance tolerance information, and that any node is able to obtain periodic performance metrics from the network. That being so, integrating the model into a composition scheme leads to the generation of a dynamic hash table HT_{vd} whereby,

- the *values* of the table represent service composition candidates currently available in the user's environment,
- the *keys* of the table represent the current performance requirement λ of the network, where $\lambda = (BW, L, D, J)$, and
- the *hash function* H maps a λ_i to a particular SCC_j, such that $H = QoS_{vd}[\lambda]$.

The completeness of HT_{vd} and the efficiency of obtaining HT_{vd} depends on whether a *push* or *pull* method is used in service advertisement. A push method (such as Chakraborty et al.'s DSC) involves nodes that are providing services by continuously broadcasting service advertisement. They are said to be pushing their services to nearby nodes. A pull method (such as VDCSP) involves a node advertising its services only when a peer node shows interest. The interested node is said to be pulling the advertisement to itself. A push-based technique leads to a largely complete HT_{vd}, because brokers of a composition continuously receive notices of new or modified services. However, this technique is inefficient: service advertisements are received whether they are desired or not. This leads to unnecessary depletion of node resources (battery and processing power). A pull-based composition is more efficient because the broker fills HT_{vd} with only the services it is explicitly interested in at the moment. The result, however, is an incomplete HT_{vd}, because, after initial composition, the broker is unaware of new or modified services in the network. The necessary adaptation requires additional work, up to and including complete recomposition.

In the following section, among other analysis, we examine whether this additional work needed in a pull technique leads to a more efficient and cost-effective solution than the resource-heavy push technique.

4 Simulation and Analysis

This section evaluates the performance of our VDCSP in varying service densities, mobility, and topologies. The section also evaluates VDCSPs ability to gracefully degrade and upgrade in postcomposition adaptation. We compare our results to those of Chakraborty et al.'s [3] push-based DSC protocol. The comparison is an effort to identify the efficiency gain in message usage and composition time, as well as to determine the reactions to varying mobility and scaling.

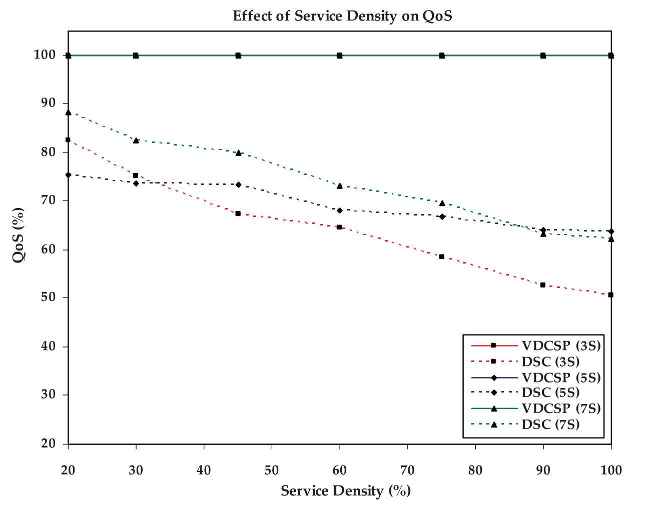


Fig. 6. QoS with respect to service density for composition lengths 3, 5, 7.

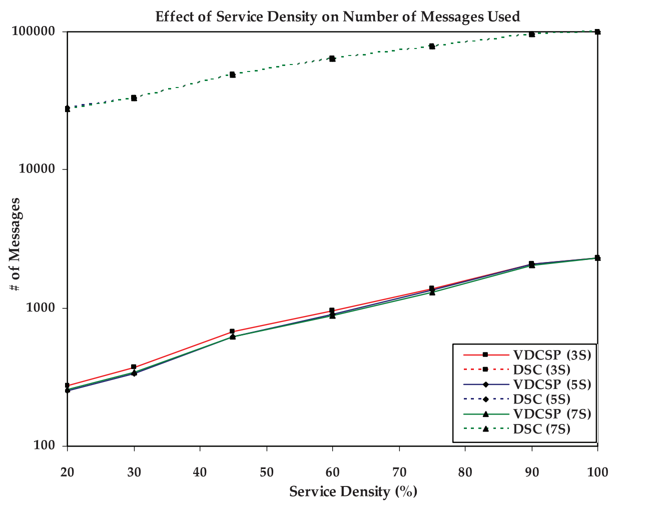


Fig. 7. Number of messages consumed with respect to service density for composition lengths 3, 5, 7.

4.1 Simulation Parameters and Metrics

We built a MANET and implemented both VDCSP and DSC protocols using J-SIM [43], a well-known simulation environment. We used an area of 30×30 m containing 25 nodes with a transmission range of 30 m. We set mobility to follow a random-way-point model with a minimum speed of 1 m/s, a maximum speed of 3 m/s, and a 5 s stoppage time. All broadcasts follow a strict bounded hop count of 1. The DSC service advertising intervals are set to 5 s. The simulations are intended to reflect a mobile device making 100 requests over a 24-hour period. The simulation was repeated for composite lengths of 3, 5, and 7, and service densities of 20-100 percent. We define service density as the percentage of nodes containing one of the atomic services required in the composition. For each simulation, we identify the optimal QoS achieved; a normalized QoS value to represent how close the actual formed composition comes to the best TCspec match currently present in the network. We also identify the amount of time taken and the number of messages consumed.

Moreover, it should be noted that since the derivation of similarity functions and weight are out of the scope of this paper, within the simulation we have utilized predetermined similarity values to focus instead on the performance of the protocols.

4.2 QoS Analysis

From Fig. 6, results indicate that, compared to DSC, VDCSP provides an average increase in QoS of 55, 44, and 35 percent when composing three, five, and seven services, respectively. VDCSP is able to provide QoS above 99 percent for all service densities in Fig. 6. DSC does not fare well, mainly because it contains no mechanism to distinguish a poor service from a better service. The QoS performance of DSC simply reflects the average in the networks, a metric that drops as service density rises.

4.3 Cost Analysis

Per composition, the number of messages used by VDCSP (Fig. 7) is an average of 62,750 messages (98 percent) lower than DSC. See Fig. 7 for the overlap in DSC curves. Average VDCSP composition time decreases by 15.39 seconds (74 percent) when compared to DSC (Fig. 8). The considerable drop in messages is because DSC requires nodes to advertise their services at periodic intervals. DSC also requires the broker election, service discovery, and service aggregation phases to be sequential, whereas VDCSP solves the problem in a single distCSP. The initial spike of DSC for composition lengths 5 and 7 (Fig. 8) can be attributed to the fact that DSC makes futile attempts to discover services that do not exist, whereas VDCSP, by finding the "best" composition possible, is able to quickly identify whether a desired service exists or not.

The periodic service advertisements of DSC place a heavy cost on the approach. In supplemental Section 2, available online, we investigate the effects of increasing the service advertisement interval to minimize message cost.

4.4 Effects of Scaling

To examine the effects of scaling on the performance of each protocol, we utilize the parameters shown in Table 1, but hold service density at 50 percent, and composition length at three services. In addition, we now gradually increase the

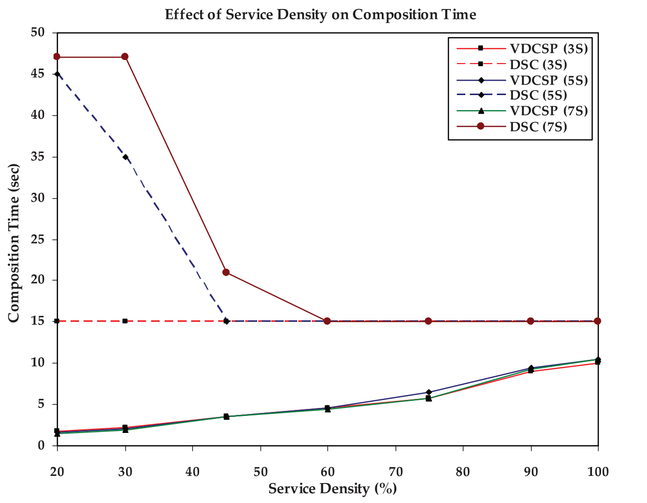


Fig. 8. Composition time with respect to service density for composition lengths 3, 5, 7.

TABLE 1
Simulation Parameter Summary

Duration	100 requests over 24 hour period
Space(x,y)	30×30 m
Transmission range	30 m
No. of nodes	25
Advertisement interval (DSC)	5 sec
Advertisement lifetime (DSC)	7 sec
Control hop count	1
Mobility	Random-way-point model with a minimum speed of 1m/s, a maximum speed of 3m/s, and a 5s stoppage time
Initial topology	Grid topology with nodes equally spaced out in (x,y)
Composite lengths	3, 5, and 7
Service density	20 to 100%

dimension area D of the experiment and the number of nodes N involved: $D = 30 + 6 \times (\sqrt{N} - 5)$.

Results show that both protocols suffer from scalability issues. The effect of scaling on VDCSP proves to be detrimental to the achieved level of QoS (Fig. 9). As the dimension area surpasses the transmission range of a node, the performance begins a linear decrease (for example, 45 to 65 nodes in Fig. 9). This can be attributed to the lack of discovery of potential services both by bounded broadcast and node disconnect during the negotiation process. It also leads to an increase in composition time (Fig. 11), but only a small increase in the number of messages (Fig. 10), which taper down as the dimension continues to grow. The latter can be attributed to the higher likelihood that more nodes are out of range of a given node as the dimension increases, resulting in negotiations between fewer nodes. The QoS performance of DSC remains unchanged (Fig. 9) and always achieves a level that matches the network average. This decreases as the number of nodes increases because of the continuous broadcasting and the protocol's heavy discovery mechanism. The cost in messages required places a far too heavy burden on the network to be practical.

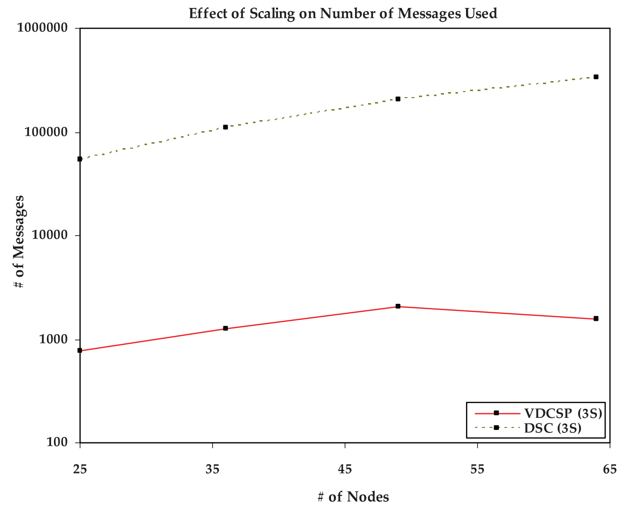


Fig. 10. Number of messages consumed with respect to scaling number of nodes and dimension.

4.5 Effects of Mobility

To examine the effects of mobility on the two protocols, we again utilize the parameters shown in Table 1, hold the service density at 50 percent, and the composition length at three services. However, we now increase the number of nodes to 49, and the dimension to 42×42 m. Mobility continues to follow a random-way-point model, but now gradually increases, with no stoppage time, from 0 to 10 m/s.

Results show that DSC's broadcast-based discovery mechanism is resilient to mobility. DSC continuously discovers the QoS level that matches the network average (Fig. 12). But VDCSP's pull-based mechanism suffers as mobility causes an increase in the number of nodes that are outside the initiating node's range. As a result, fewer nodes participate and spend less time solving the DistCSP (Figs. 13 and 14). An interesting observation occurs when nodes

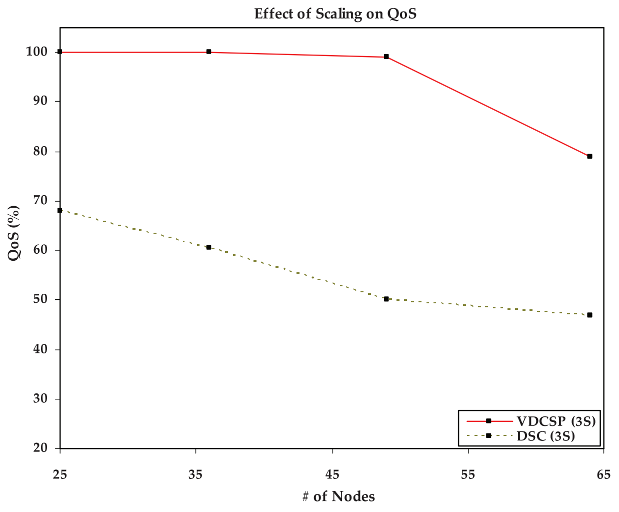


Fig. 9. QoS with respect to scaling number of nodes and dimension.

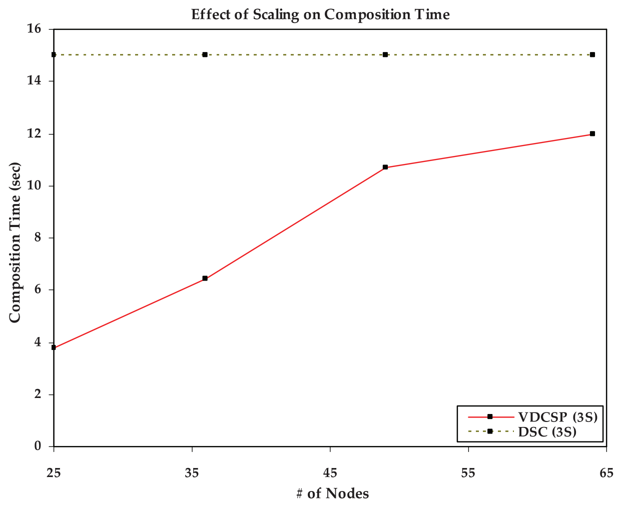


Fig. 11. Composition time with respect to scaling number of nodes and dimension.

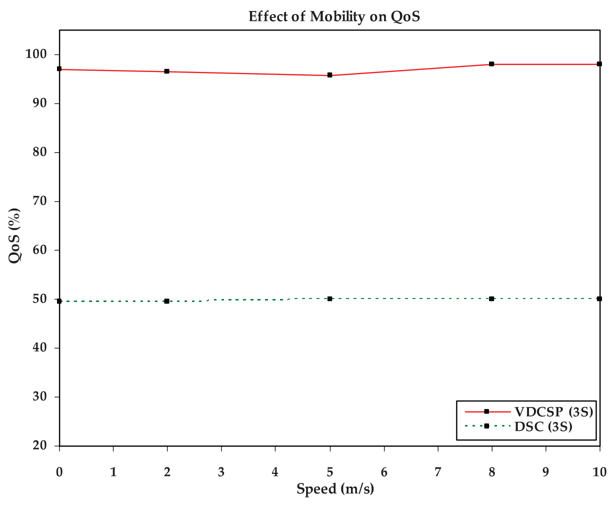


Fig. 12. QoS with respect to mobility speed.

move in and out of the initiating nodes' range at a quick enough rate to be discovered and participate in the DistCSP. There is a subsequent rise in QoS, composition time and number of messages used. Although DSC is more resilient than VDCSP in this experiment, the cost far outweighs the benefit. Once again, the number of messages required places a far too heavy burden on the network to be practical.

4.6 Postcomposition Adaptation

In this section, we modify DSC to include our QoS model $QoS_{vd}[PR]$ in order to analyze the effect of service advertisement techniques on postcomposition adaptation.

Using the parameters shown in Table 1, our simulations represents a mobile device entering an environment and forming a virtual device, followed by 100 simulated changes in bandwidth, delay, loss, and jitter. This forces the virtual device to make 100 corresponding adaptations.

From Fig. 15, results indicate that both VDCSP and DSC are able to continuously discover the optimal adaptation (with lines overlapping at 100 percent) for varying service densities and composition lengths. This is expected as the two protocols are using identical QoS models. The differences come in the cost of achieving adaptation.

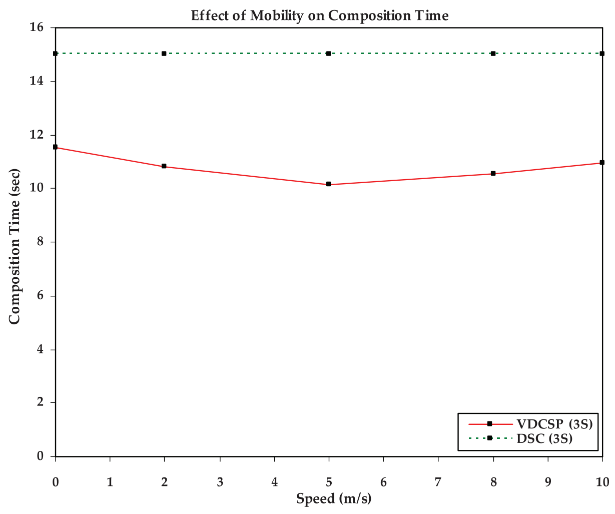


Fig. 14. Composition time with respect to mobility speed.

Per adaptation, the number of messages used by VDCSP (Fig. 16) is an average of 89 percent less than DSC. However, DSC's average composition time decreases by 3.05 seconds when compared to VDCSP. Fig. 17 shows that DSC holds at about 0.15 seconds for all service densities and composition lengths. The results provide a visual example of the tradeoff between the push and pull methods. As DSC uses a push method, the broker is continuously receiving periodic service advertisements from nodes in the vicinity. Fig. 16 shows that the higher the service density (the more advertising nodes), the higher the number of messages. The benefits of this technique are clearly shown in Fig. 17: using DSC, the broker's HT_{vd} requires hardly any time at all for the virtual device to adapt. By contrast, the time required for VDCSP to adapt is directly related to service density and composition length. As the performance metrics of the network change, recomposition is necessary in order for the broker to acquire sufficient information to initiate the optimal adaptation. This is also seen in Fig. 16, where, using a pull method, the process of composition is significantly more efficient than DSC.

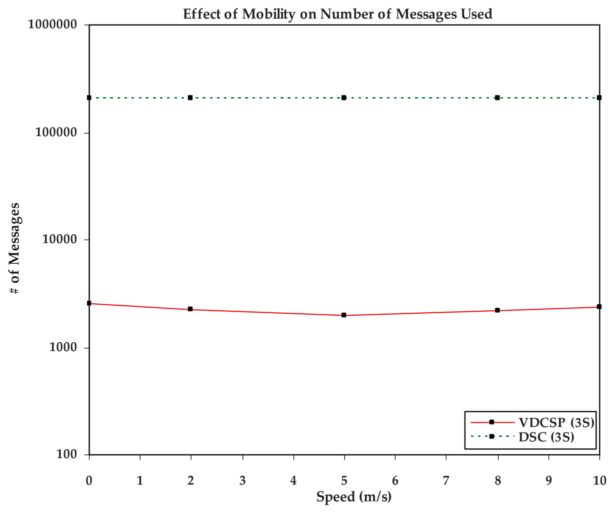


Fig. 13. Number of messages consumed with respect to mobility speed.

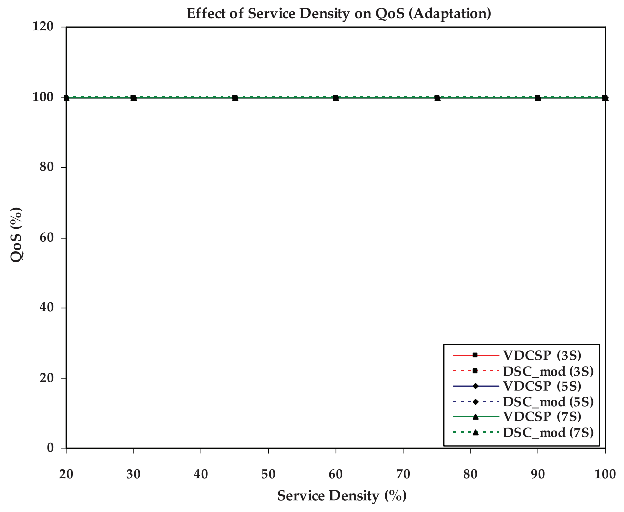


Fig. 15. QoS (adaptation) with respect to service density for composition lengths 3, 5, 7.

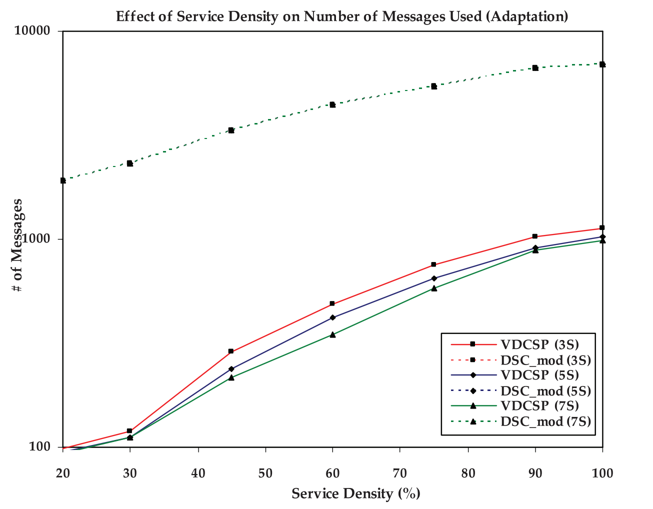


Fig. 16. Number of messages consumed (adaptation) with respect to service density for composition lengths 3, 5, 7.

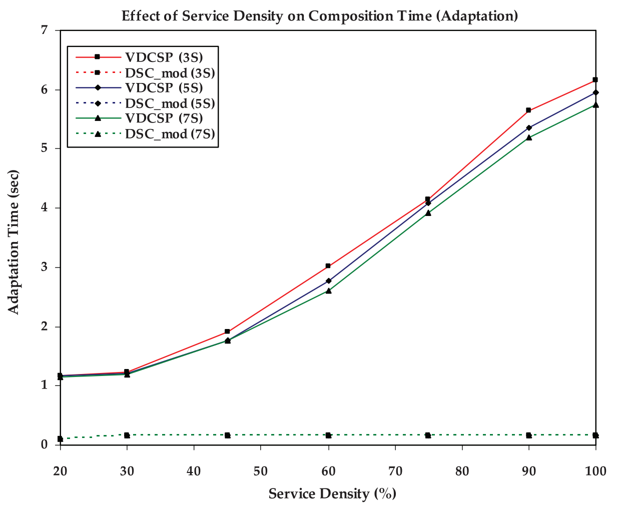


Fig. 17. Composition time (adaptation) with respect to service density for composition lengths 3, 5, 7.

Our results show that our VDCSP protocol would benefit from small amounts of pushed service advertisements. A hybrid push/pull technique would potentially offer better results. Rather than continuous broadcasts, the technique could involve a single broadcast when a service is introduced, modified, or about to be deleted. This would allow the broker to keep a more complete HT_{vd}, minimizing the frequency of recomposition while avoiding heavy resource loss.

5 Conclusion

In this paper, we have presented a distCSP model for task-based, QoS-aware, virtual device composition in MANETs, and an asynchronous backtracking algorithm for solving the distCSP. The model provides a method of virtual device composition without continuous broadcast-based service advertisements, and without the need for agents to directly divulge specific information about their domain and constraints. We have also presented a network-performance-influenced QoS model for the graceful degradation and upgradation of a virtual device postcomposition. Through simulation, we have shown that our method is effective in achieving high QoS. compositions without the unnecessary depletion of node resources. We have also discussed and simulated some drawbacks of the technique in the context of push- and pull-based service advertisement.

Future work will investigate the design of a hybrid push/pull technique with multihop broadcasting. Multihop composition was not considered in this paper such that only single hops are necessary in discovering devices within the vicinity of the user. However, reconciliation services (e.g., codec transformation services) are not necessarily required to be near the user, and the discovery of such services could potentially benefit from multi-hop broadcasts. Furthermore, the lessons learned will be applied as we experiment with more constrained upgrade policies to avoid the frequent changes caused by dynamic network conditions.

Acknowledgments

This work was supported in part by a grant from Natural Sciences and Engineering Research Council of Canada.

References

[1] W. Buxton, "Less Is More (More or Less)," *Invisible Future: The Seamless Integration of Technology in Everyday Life*, P. Denning, ed., pp. 145-179, McGraw Hill, 2001.

[2] M. Merabti, P. Fergus, O. Abuelma'atti, H. Yu, and C. Judice, "Managing Distributed Networked Appliances in Home Networks," *Proc. IEEE*, Special Issue on Recent Advances in Distributed Multimedia Comm., vol. 96, no. 1, pp. 166-185, Jan. 2008.

[3] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, "Toward Distributed Service Discovery in Pervasive Computing Environments," *IEEE Trans. Mobile Computing*, vol. 5, no. 2, pp. 97-112, Feb. 2006.

[4] G. Prochart, R. Weiss, R. Schmid, and G. Kaefer, "Fuzzy-Based Support for Service Composition in Mobile Ad Hoc Networks," *Proc. IEEE Int'l Conf. Pervasive Services*, pp. 379-384, 2007.

[5] S. Jiang, Y. Xue, and D. Schmidt, "Minimum Disruption Service Composition and Recovery over Mobile Ad Hoc Networks," *Proc. Fourth Ann. Int'l Conf. Mobile and Ubiquitous Systems: Networking and Services*, pp. 1-8, 2007.

[6] M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara, "The Distributed Constraint Satisfaction Problem: Formalization and Algorithms," *IEEE Trans. Knowledge and Data Eng.*, vol. 10, no. 5, pp. 673-685, Sept./Oct. 1998.

[7] E. Karmouch and A. Nayak, "A Distributed Protocol for Virtual Device Composition in Mobile Ad Hoc Networks," *Proc. IEEE Int'l Conf. Comm. (ICC)*, 2009.

[8] E. Karmouch and A. Nayak, "A Distributed Constraint Satisfaction Problem for Virtual Device Composition in Mobile Ad Hoc Networks," *Proc. IEEE GLOBECOM*, 2009.

[9] F. Casati, M.C. Shan, and D. Georgakopoulos, "E-Services - Guest Editorial," *The Int'l J. Very Large Databases*, vol. 10, no. 1, p. 1, 2001.

[10] A. Lazcano, "WISE: Process-Based E-Commerce," *IEEE Data Eng. Bull.*, Special Issue on Infrastructure for Advanced e-Services, vol. 24, no. 1, pp. 46-51, Mar. 2001.

[11] C. Thompson, P. Pazandak, V. Vasudevan, F. Manola, G. Hansen, and T. Bannon, "Intermediary Architecture: Interposing Middleware Object Services between Web Client and Server," *Proc. Workshop Compositional Software Architectures*, 1998.

[12] R.H. Katz, E.A. Brewer, and Z.M. Mao, "Fault-Tolerant, Scalable, Wide-Area Internet Service Composition," Technical Report UCB/CSD-1-1129, CS Division, EECS Dept., Univ. of California at Berkeley, Jan. 2001.

[13] J.N. Kok and K. Sere, "Distributed Service Composition," Technical Report no. 256, Turku Centre for Computer Science, Finland, Mar. 1999.

[14] A. Brocco and B. Hirsbrunner, "Service Provisioning for a Next-Generation Adaptive Grid," *Int'l J. Parallel, Emergent and Distributed Systems*, vol. 26, no. 1, pp. 85-106, Apr. 2011.

[15] P. Queloz and A. Villazon, "Composition of Services with Mobile Code," *Proc. Third Int'l Symp. Mobile Agents*, 1999.

[16] D. Chakraborty and A. Joshi, "Dynamic Service Composition: State-of-the-Art and Research Directions," Technical Report TR-CS-01-19, Univ. of Maryland Baltimore County, Dec. 2001.
[17] Y. Feng, J. Cao, I. Chuen, H. Lau, Z. Ming, and J. Kee-Yin Ng, "A Component-Level Self-Configuring Personal Agent Platform for Pervasive Computing," *Int'l J. Parallel, Emergent and Distributed Systems,* vol. 26, no. 3, pp. 223-238, June 2011.
[18] WSDL, "Web Services Description Language 1.1," http://www.w3.org/TR/wsdl, 2012.
[19] DARPA, "Agent Markup Language for Services Specification Draft 0.5," http://www.daml.org/services/daml-s/2001/05/, 2012.
[20] W3C, "OWL Web Ontology Language Overview," http://www.w3.org/TR/owl-features/, 2012.
[21] WSFL, "Web Services Flow Language," http://xml.coverpages.org/wsfl.html, 2012.
[22] IBM, "Business Process Execution Language for Web Services Version 1.1," http://www.ibm.com/developerworks/library/specification/ws-bpel/, 2012.
[23] K. Erol, J. Hendler, and D. Nau, "HTN Planning: Complexity and Expressivity," *Proc. Int'l Conf. Artificial Intelligence,* 1994.
[24] M. Paolucci, A. Ankolekar, N. Srinivasan, and K. Sycara, "The Daml-s Virtual Machine," *Proc. Second Int'l Semantic Web Conf.,* 2003.
[25] K. Arnold, B. Osullivan, R.W. Scheifler, J. Waldo, and A. Wollrath, *The Jini Specification.* The Jini Technology Series. Addison-Wesley, June 1999.
[26] The Salutation Consortium, Inc., "Salutation Architecture Specification (Part 1) Version 2.1," http://systems.cs.colorado.edu/grunwald/MobileComputing/Papers/Salutation/Sa20e1a21.pdf, 2012.
[27] R. John, "UPnP, Jini and Salutaion - A Look at Some popular Coordination Frameworks for Future Network Devices," technical report, California Software Labs, 1999.
[28] E. Guttman, C. Perkins, and J. Veizades, "Service Location Protocol," RFC 2165, June 1997.
[29] S.E. Czerwinski, B.Y. Zhao, T.D. Hodes, A.D. Joseph, and R.H. Katz, "An Architecture for a Secure Service Discovery Service," *Proc. Fifth Int'l Conf. Mobile Computing and Networks,* 1999.
[30] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan, "Adaptive and Dynamic Service Composition in eFlow," Technical Report HPL-200039, Software Technology Laboratory, Palo Alto, California, Mar. 2000.
[31] S. Helal, N. Desai, and C.L. Konark, "A Service Discovery and Delivery Protocol for Ad-Hoc Networks," *Proc. IEEE Third Conf. Wireless Comm. Networks,* 2003.
[32] D. Tang, C. Chang, K. Tanaka, and M. Baker, "Resource Discovery in Ad Hoc Networks," Technical Report CSL-TR-98-769, Stanford Univ., Aug. 1998.
[33] S. Han and Y. Zhang, "Design and Implementation of Service Composition Protocol Based on DSR," *Proc. Int'l Conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT),* pp. 323-328, Dec. 2010.
[34] J. Wang, "Exploiting Mobility Prediction for Dependable Service Composition in Wireless Mobile Ad Hoc Networks," *IEEE Trans. Services Computing,* vol. 4, no. 1, pp. 44-55, Jan. 2010.
[35] Bluetooth Specification, http://www.bluetooth.org./, 2012.
[36] M. Klemettinen, *Enabling Technologies for Mobile Services: The MobiLife Book.* Wiley, 2007.
[37] R. Want, T. Pering, S. Sud, and B. Rosario, "Dynamic Composable Computing," *Proc. Ninth Workshop Mobile Computing Systems and Applications,* pp. 17-21, 2008.
[38] C. Namman, A. Mingkhwan, O. Abuelma'atti, and M. Merabti, "The Flexible Service Composition Framework for Networked Appliances," *Proc. Int'l Conf. Innovations in Information Technology,* pp. 233-237, 2007.
[39] R. Thiagarajan and M. Stumptner, "Service Composition with Consistency-Based Matchmaking: A CSP-Based Approach," *Proc. Fifth European Conf. Web Services,* pp. 23-32, Nov. 2007.
[40] I. Paik, D. Maruyama, and M.N. Huhns, "A Framework for Intelligent Web Services: Combined HTN and CSP Approach," *Proc. Int'l Conf. Web Services (ICWS),* pp. 959-962, Sept. 2006.
[41] A. Joshi and D. Chakraborty, "GSD: A Novel Group-Based Service Discovery Protocol for MANETS," *Proc. Fourth Int'l Workshop Mobile and Wireless Comm. Networks,* pp. 140-144, 2002.
[42] M. Perttunen, M. Jurmu, and J. Riekki, "A QoS Model for Task-Based Service Composition," *Proc. Fourth Int'l Workshop Managing Ubiquitous Comm. and Services,* pp. 11-30, 2007.
[43] J-Sim, "Home (J-Sim Official)," J-Sim, http://sites.google.com/site/jsimofficial/, Jan. 2005.

Eric Karmouch received the BSc degree in computer science from the University of Ottawa in 2004 and the MSc degree in computer science from Queen's University in 2006. He is currently working toward the PhD degree with the School of Electrical Engineering & Computer Science at the University of Ottawa. His research interests include distributed, mobile, and pervasive computing, and ambient intelligence. He is a member of the IEEE and the IEEE Computer Society.

Amiya Nayak received the BMath degree in computer science and combinatorics and optimization from the University of Waterloo, Canada, in 1981, and the PhD degree in systems and computer engineering from Carleton University, Canada, in 1991. He has more than 17 years of industrial experience, working at CMC Electronics, Defence Research Establishment Ottawa, EER Systems and Nortel Networks, in software engineering, avionics and navigation systems, simulation and system level performance analysis. He is in the Editorial Board of *IEEE Transactions on parallel and distributed systems, International Journal of parallel, emergent and distributed systems, International Journal of computers and applications, International Journal of computer information technology and engineering, International Journal of computing and information science, International Journal of autonomic computing, and EURASIP Journal on wireless communications and networking*. Currently, he is a full professor at the School of Electrical Engineering & Computer Science at the University of Ottawa, Canada. His research interests are in the areas of mobile ad hoc and sensor networks, fault tolerance, and distributed systems/algorithms. He is a senior member of the IEEE.