CiAN: A Workflow Engine for MANETs

Rohan Sen, Gruia-Catalin Roman, and Christopher Gill

Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045, One Brookings Drive, St. Louis, MO 63130, U.S.A.
`{rohan.sen,roman,cdgill}@wustl.edu`

Abstract. The practice of using workflows to model complex activities in stable networks is commonplace and is supported by many commercially available workflow management systems (WfMSs). However, the use of workflows to model collaborative activities in mobile environments, while possible at the model level, has not gained traction due to the lack of a suitable WfMS for mobile networks and devices. This paper seeks to address this need. We present CiAN, a choreography-based workflow engine that is designed with MANETs in mind. We describe the design, architecture, and communication protocols used by CiAN as well as its implementation using Java. An evaluation of the communication protocol used to coordinate among various workflow participants across MANETs is also presented.

1 Introduction

Workflows can be conceptualized as a set of related tasks that are arranged according to a specific order and structure to accomplish a higher level goal in a collaborative manner. Workflows are commonly represented and specified in terms of graphs or petri-nets [23]. Software systems that execute these workflow specifications are called Workflow Management Systems (WfMSs). In the current state of the art, WfMSs such as ActiveBPEL [9], Oracle Workflow Engine [18], Biztalk [7], etc. operate across wired networks and execute workflows that encode complex business processes such as insurance claims processing, inventory control, loan approvals, among others.

A WfMS has two main functions: assigning tasks in the workflow to suitable hosts and subsequently invoking them in the correct order, passing any data or notifications between them as necessary. The performance of all the tasks by multiple participants collectively accomplishes the collaborative activity specified by the workflow. Current designs for WfMSs reflect the stable and reliable environment in which they operate. The architecture of these systems are centralized and interactions with the various distributed components are typically synchronous calls made "just-in-time".

In this paper, we describe the design of a WfMS targeted to mobile settings. Our work is motivated by the fact that while the workflow model is robust enough to describe more expansive forms of collaborations (including collaborations involving both humans and software in the physical world), it is not in

D. Lea and G. Zavattaro (Eds.): COORDINATION 2008, LNCS 5052, pp. 280–295, 2008.

widespread use due to the lack of a suitable WfMS to execute such workflows. A mobile WfMS that can operate over a mobile ad hoc network (MANET) can be used as a general purpose coordination mechanism for the activities of workers at a remote outdoor construction site, management of emergency responders in the event of a toxic chemical spill, or directing the activities of a geological survey team where setting up a traditional WfMS over a temporary LAN, even if possible, is not desirable.

However, developing a WfMS for MANETs has several implications, the most significant of which is the paradigm shift from centralized management to a distributed management scheme. In addition, appropriate communication and coordination protocols need to be developed so that participants can interact over a dynamic and fragmented network. CiAN, which stands for Collaboration in Ad hoc Networks, is a clean sheet approach to building a WfMS that is flexible enough to operate across a MANET. CiAN is designed from the ground up to function in a choreographed manner, i.e., in a manner that does not require a central coordinating entity. Novel features of CiAN include a distributed management system that functions at the level of granularity of a single task, a communication protocol that combines publish-subscribe, store-and-forward, and content-based routing to foster communication across the MANET between various hosts performing the workflow, and an ability to adapt the workflow execution according to changes in the context in which the execution takes place.

2 Background

Before we present the features of our system in detail, we describe precisely our target environment and the differences between operating in a choreographed manner as opposed to the more commonplace orchestrated manner.

For CiAN, we assume that there exists a group of human users, each of whom is equipped with a relatively powerful mobile computing device (in the remainder of the paper we refer to the device and user collectively as a *host*). We assume that all hosts are co-located initially but may separate once the workflow execution has begun. Since the devices are carried on the person of the users, we assume that the devices are physically mobile and that their motion pattern is the same as their associated user. The devices are capable of communicating with each other using 802.11b/g/n radios when they are within communication range of each other. However, such *windows of communication* (the intervals of time during which a pair of hosts are within range) may be transient due to the mobility of the associated human user.

Each host that participates in the execution of a workflow provides: (1) A name, assumed to be unique in the network, (2) A schedule with entries that indicate when it is not available. Each entry consists of a start time, location at the start time, end time, and location at the end time. When hosts are assigned tasks, they add them to their schedule so that they are not assigned additional tasks that conflict, and (3) A list of services offered. This list includes software services on the mobile devices and the associated user's capabilities, e.g., a metal

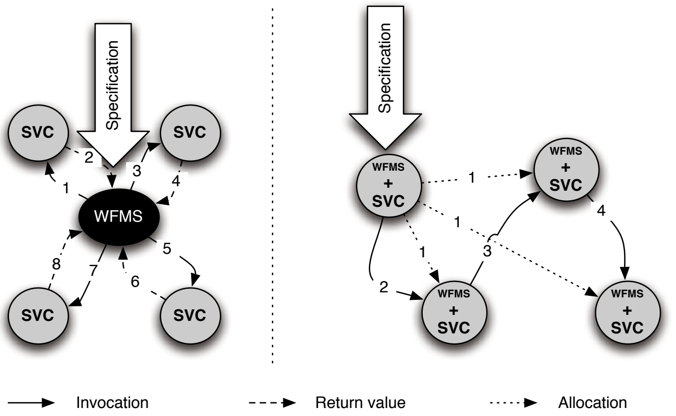

Fig. 1. Orchestration vs. Choreography (SVC = Service & WFMS = Workflow Mgmt. System.

worker may have welding capabilities. We assume that each host maintains a local knowledge base [12] in which it keeps information about other hosts in the network. Initially the knowledge base contains information only about the local host. However, over time, the knowledge base is populated with information about other hosts when pairs of hosts are within communication range of each other via a gossiping protocol. The contents of the knowledge base can be queried by other components of the middleware. It should be noted that due to all hosts being co-located initially, each host has knowledge of all others in the network. However, future updates to host knowledge are dispersed via gossiping which may lead to asymmetric information in the network.

Since there is no central coordinating entity in this environment, all management functions must be handled in a distributed manner. This requires the execution model to be *choreographed*. In choreography, the responsibility for executing the workflow is divided up a priori by an allocation algorithm (not covered in this paper. Please refer to [25]). The various participants then interact with each other directly via a peer-to-peer model using pre-established standardized protocols. This is in sharp contrast to the more common *orchestrated* architecture where a centralized entity is responsible for executing the entire workflow and synchronously invokes services (in workflow order) to complete tasks. The differences between these approaches are shown pictorially in Figure 1.

The following section describes our design for a choreography-based WfMS along with the communication protocols for communicating with various components across the MANET.

3 System Design

According to the W3C definition, choreography "defines re-usable common rules that govern the ordering of exchanged messages, and the provisioning patterns of collaborative behavior, as agreed upon between two or more interacting participants.". In the context of our WfMS, this translates to the allocation of tasks to hosts (which in combination with the workflow structure describes the agreed upon collaboration patterns among participating hosts) while the execution engine is responsible for implementing the rules and protocols governing the exchange of messages. To keep these two concerns separate, CiAN operates in two modes: (1) *planning* - which is used to allocate tasks in the workflow and (2) *standard* - which is used by the hosts whose responsibility is to perform the tasks that have been allocated to it. This paper focusses on the standard mode. We include a brief presentation of the planning mode for completeness.

3.1 CiAN in Planning Mode

The Planning Mode of CiAN is responsible for implementing a scheme to inform each participating host of its role in the overall workflow. If the allocation of tasks is being done centrally, a single host operates in planning mode (hereinafter referred to simply as the *planning host*) and runs a centralized allocation algorithm [13] which allocates individual tasks to hosts. If the allocation of tasks is being done in a distributed manner [25], then several hosts run in planning mode. The host that initiates the workflow is responsible for fragmenting the workflow and passing it to the other hosts running in planning mode along with a set of rules for task allocation. For the purposes of our discussion, we will assume that the allocation process is centralized followed by a distributed, choreographed execution. It should also be noted that a host can run the planning and standard modes of CiAN simultaneously, if it so desires.

The planning host allocates each task in the workflow to a *suitable* host, where a suitable host is defined as a host whose capabilities are a superset of the capability requirements of the task, and whose motion pattern allows it to be at the location at which the task needs to be performed at the time it needs to be performed. Figure 2 shows the system architecture on the planning host. An external application injects the workflow specification into the planning system by way of the `Planner`. The `Planner` passes this specification to the `Allocator`, which runs an appropriate *allocation algorithm* (e.g., [13] or [25]) to determine the hosts that are assigned to each task in the workflow. It then annotates the specification with these allocations and returns it to the `Planner`. The `Planner` then feeds the specification to the `Route Information` unit, which augments the specification with metadata (used for data routing - described later in this section). This augmented specification is then returned to the `Planner` which now forwards it to the `Specification Disbursement Policy` module, which breaks the workflow into its constituent tasks and sends each task specification to the host that has been allocated to perform it using the `Communication Middleware`. Each task specification sent out includes (1) the input edges to the task, their merging and synchronization pattern [27], and the tasks at the source of the edges, (2) the service that must be invoked for that task, and (3) the output edges to the task, their splitting and synchronization pattern [27], and the tasks at the sinks of the edges.

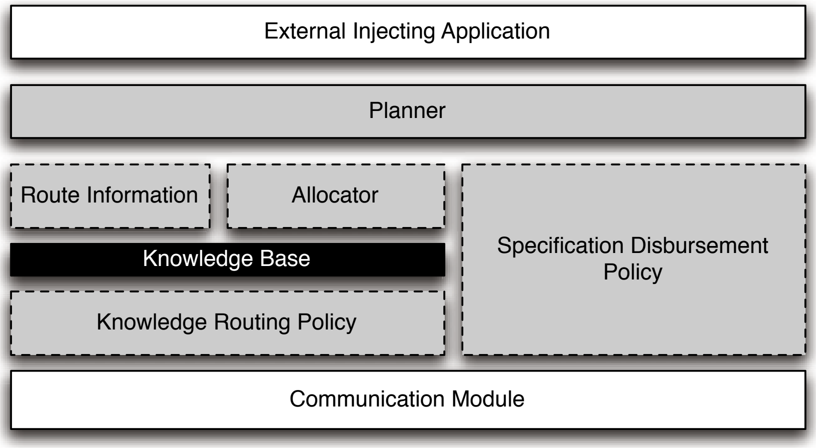

Fig. 2. CiAN planning architecture

There are a few points to note in the figure. The components with dotted borders are interfaces, i.e., they can be realized by alternate policies as long as they meet the interface requirements. One simple example of this is the replacement of the `Allocator` which implements a centralized algorithm in our description this far with one that sets up a distributed allocation policy. The `Allocator` uses the schedule and service list provided by each host (stored in the `Knowledge Base` as described in Section 2) to determine the allocation of the tasks to hosts based on their capabilities and motion constraints. Recall that since hosts are co-located initially, the planning host has access to information about all participating hosts.

3.2 CiAN in Standard Mode

The Standard Mode of CiAN is responsible for managing the choreographed execution of the workflow on individual hosts and then disbursing results to the hosts that are responsible for executing subsequent tasks. At a high level, the Standard Mode on a given host works as follows: (1) It waits for a task to be allocated to the host on which it is executing. (2) When a task is allocated, it receives the specification for that task and installs it within the system and goes back to waiting (either on inputs to the task it has installed or additional task allocations). (3) If an input to an installed task is received, it runs the input synchronization logic for that task (see Figure 4). If the logic is satisfied, the values received are passed to the task for execution. If not, then additional inputs may be required and the system waits for these. (4) When the task execution has been completed, it runs the output synchronization logic for that task and transmits the values to the tasks at the sinks of the outgoing edges. We now describe this process in detail.

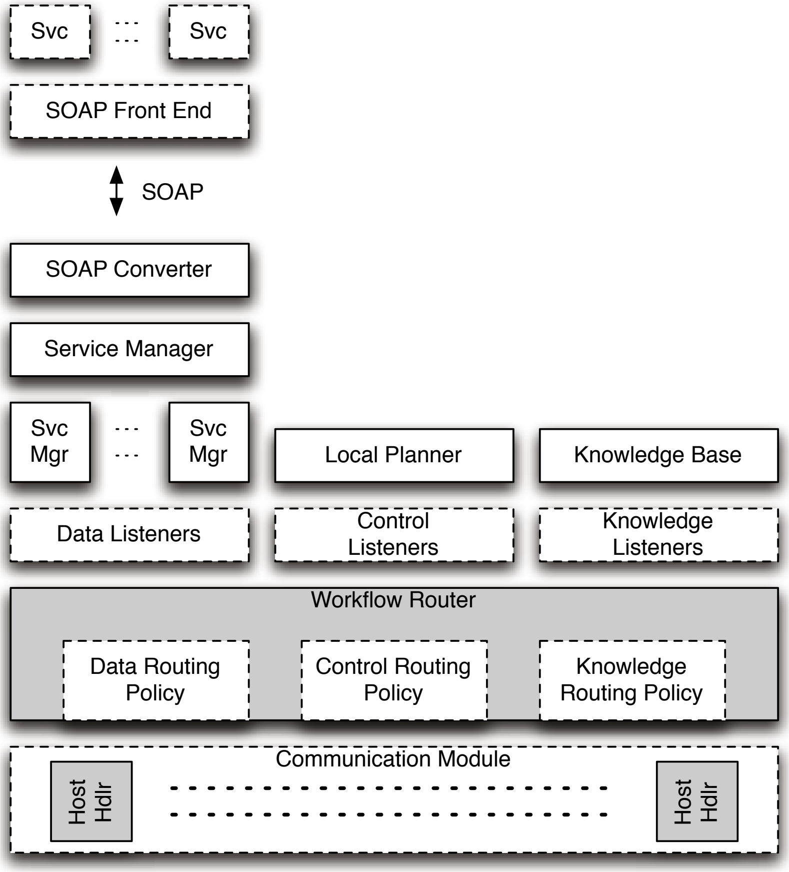

Fig. 3. CiAN runtime system architecture

The architecture of CiAN in standard mode is shown in Figure 3. When the task specification arrives, the `Communication Module` passes it to the `Workflow Router`. The arrival of the specification is regarded as a *control message*, so it is given to the `Control Routing Policy` module of the `Workflow Router`, which in turn notifies any `Control Listeners` that may be listening for these messages. The default `Control Listener` parses the task specification and creates a `Service Manager` for the task. The `Service Manager` contains the input synchronization and output synchronization logic mentioned above, which are parametrized according to the information in the task specification received. For example, if a task has three incoming edges with AND join semantics, the input synchronization logic would not be satisfied until it had received values from all three edges. The `Service Manager` creates *subscriptions* for each of its inputs, which is a request for data generated by its preceding tasks (we will cover subscriptions later in this section). At this point a task is waiting on its inputs before it can start executing.

The first task in any workflow by definition does not have any inputs, and hence can start executing immediately. The `Service Manager` must invoke the service that performs the activity associated with the task. In CiAN, we assume that all services can be accessed via SOAP calls. The `Service Manager` calls the `SOAP Converter` which converts the service call into a SOAP request. This is then handed off to a `SOAP Front End` which receives the request and routes it to the appropriate service. The response from the service is translated into a SOAP response and returned to the `Service Manager` via the same route. At this stage, the `Service Manager` executes output synchronization logic. If the logic is satisfied, it passes the data to the `Data Policy` of the `Workflow Router` which then transmits it to the host(s) that is(are) responsible for performing the task(s) immediately following the first task. These tasks wait on their inputs and execute once all the inputs are available. Execution continues until the last task in the workflow is executed. This is what creates the choreographed form of workflow management in CiAN. Two points to note in addition: (1) The pluggable components in the middleware allow easy extensibility and more importantly allow the middleware to be customized as per the specific requirements of the domain and (2) The SOAP interface to the services allows compatibility with Web services.

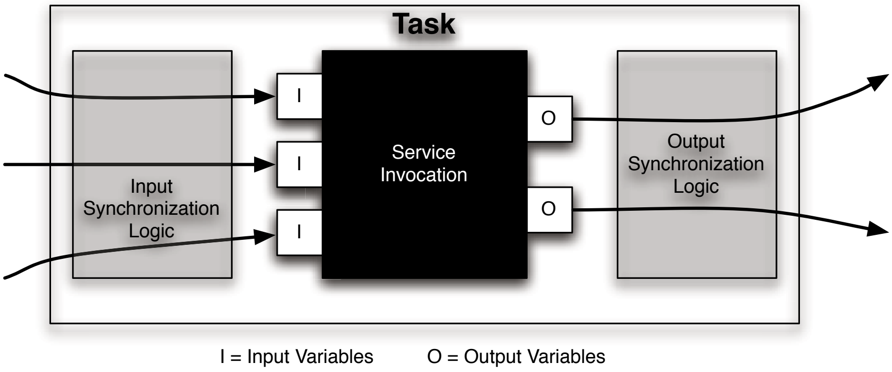

Fig. 4. Task and synchronization logic

3.3 Communication in MANETs

Thus far, we have described how individual tasks get executed but have not covered how the coordination of the hosts performing the tasks is handled. At the coordination and communication layer, there are two key issues: (1) The hosts are connected by a MANET, whose topology evolves rapidly over time and where unpredictable disconnections are commonplace, making it difficult to maintain long lasting routes between host pairs. (2) The workflow specification indicates which *task* a result must be delivered to or obtained from but not the *host* that is executing those tasks. We addressed these issues via a publish-subscribe-like protocol that opportunistically gossips data and subscriptions among hosts when they are directly connected with each other. The scheme is described in detail below.

The `Communication Module` on each host transmits a beacon periodically. When the `Communication Module` on another host receives such a beacon, it creates a `Host Handler` for that host. The `Host Handler` tries to establish a direct connection between the hosts using TCP/IP streams. Thus, as long as the hosts are in communication range, the `Host Handler` acts as the local proxy of the remote host and handles communication between them. Since direct communication is the most reliable and inexpensive form of communication in a MANET, all information in CiAN is transmitted when two hosts are directly connected. Thus, when the `Host Handler` establishes a connection, it synchronizes the knowledge base of the two hosts using the time of acquisition of any knowledge as a tie breaker. It also sends to and receives data or subscription messages from the other host as appropriate. All data and subscription messages received are passed to the `Data Routing Policy` in the `Workflow Router`. If a data message is intended for a task on the local host, the `Data Routing Policy` passes it to the `Service Manager` of the target task. The `Service Manager` then runs the synchronization logic to see if a valid set of inputs have been received.

While this form of communication is acceptable for gossiping, it does not meet all our requirements, specifically, it provides no means for a message exchange to take place between two hosts that are never directly connected to each other. This restriction can result in critical data from one task not reaching the next. A simple solution to this problem is to simply address each message to its destination host and use a MANET routing protocol to deliver the message. However, this has two drawbacks: (1) MANET routes do not last often and are expensive to maintain, and (2) it strongly associates a task with a host, which while not desirable is preferably avoided. Our approach is instead a store and forward approach based on a routing policy we have developed. At the planning stage, we augment each task with a unique number (the metadata mentioned earlier) such that it is greater than all its parents' numbers but lesser than all its childrens' (tasks that are siblings may have numbers lesser or greater depending on the graph traversal method used). When each host receives a task spec, it assigns a number to itself that is the same as the number of the task. If multiple tasks are assigned, then it initially chooses the lowest numbered task. Once the task associated with that number has been completed, it examines the remaining set of tasks allocated to it and chooses the lowest number available. Subscriptions (generated by tasks to solicit inputs) have the number of the subscribing task, and the number of the task whose input is desired. Similarly, when a task finishes execution, the data is labeled with the generating task number and the number of the task(s) that should receive the data. The messages are routed using one of the following three schemes: *Scheme 1* - Data is routed to any host that has a number between the generating task number and the target task number or has no number in a strictly increasing fashion. Subscriptions are routed similarly but in a strictly decreasing function. Routing to a host with no number is neither a decrease nor an increase. *Scheme 2* - Data can be routed to any host that has a number between the starting task number and the target task number in a strictly increasing order. Subscriptions are routed to hosts between the target task number and the ending task number. Routing to hosts without a number is also permitted. *Scheme 3* - This scheme is identical to Scheme 1 with one exception. Any message can be routed outside the permissible range but this triggers a counter. If the message moves to a host in range (as defined by Scheme 1) before the counter expires, the counter is reset, otherwise the message is destroyed.

Scheme 1 generates the lowest number of messages in the network but is restrictive in the sense that the number of hosts that a message can be routed to is much smaller than the total number of hosts collaborating. Scheme 2 increases the permissible range but generates additional messages. Scheme 3 maintains the low range of Scheme 1 but allows limited transgressions, which represents the most favorable tradeoff between number of messages and number of hosts to which the message can be routed. The use of task numbers for routing instead of host names or IP addresses achieves the decoupling between tasks and hosts.

Thus, communication of data between a pair of hosts proceeds as follows: The `Service Manager` on the receiving host issues a subscription for the data. When the source host has finished executing the source task, the `Service Manager` on that host creates a data message which it then passes to the `Data Routing Policy`. At this stage, our publish-subscribe-like protocol takes over and gossips it using one of the schemes described above. When a subscription and its corresponding data "meet" on a host, a match is generated and the data forwarded to the subscriber using AODV [22]. When the data is received on the receiving host, it is passed to the `Service Manager` who then runs the synchronization logic and invokes the next task.

3.4 Exploiting Mobility

Mobile systems work in a physical environment and it is desirable that these systems adapt their behavior to their environment. For WfMSs, this can be achieved by the use of *selection conditions*. Each edge to a task may have one or more selection conditions with one or more associated sub-conditions. If an edge has at least one selection condition for which all its sub-conditions evaluate as true, then the edge is marked *active*, otherwise the edge is marked as *inactive*. The sub-conditions that make up the selection condition are of the form `paramname, comparator, value` where `paramname` can be the name of an edge, a parameter in the local knowledge base, or the name of a sensor. For example `sensor:velocity, >, 10m/s` tests if the velocity of the host is greater than 10m/s.

This type of support can be built through extensions to existing languages, or a new language like the XML-based CiAN Workflow Specification which we are developing (see `mobilab.cse.wustl.edu/Projects/CiAN` for more information). Due to space constraints, it is not possible to describe all the tags in the CiAN specification. A detailed explanation of all the specification features and examples is available online at `http://mobilab.cse.wustl.edu/Projects/CiAN`.

4 Evaluation

We implemented a prototype of the CiAN WfMS in Java. The calls to external services are SOAP calls. To translate between the textual representation of the input values and SOAP requests, we use kSOAP [17], a third party library. The task of invoking the service and obtaining the return value is handled by Sliver [10], a middleware developed in our lab. Sliver currently supports the invocation of Java services only. However, since the request and response are in the form of a SOAP message, CiAN can invoke services in another language by simply adding a third party SOAP front end that is capable of invoking services written in another language. In other words, CiAN can invoke any service that can be invoked via SOAP calls if an appropriate front end is provided. Hosts participating in the workflow can register their own front end with CiAN, resulting in a situation where one host runs Java services while another runs C++ services while a third might run both. Thus, CiAN is not restricted to services written in any programming language. With the addition of language specific parsers, CiAN can also support any workflow specification language.

In addition to our implementation, we measured the performance of our publish-subscribe-like protocol to exchange data among hosts across a MANET. This is the most crucial piece of the CiAN WfMS and its primary potential bottleneck. Invocations of services to perform tasks do not take much time or resources as they are local service calls. Rather, transmitting results and receiving inputs takes significantly more time due to the communication delays. We refer to the time when tasks are being invoked and performed as *relevant time* and the time spent getting the results of one task to another as *overhead time.* Note that the system may be idle during relevant time periods (especially if the task involves a human user doing some physical chore), but it is not considered wasted time as a task is actually being performed. In our experiments, we focused on the overhead of our system since relevant time cannot be reduced due to the task duration limits set in the workflow specification.

We simulated the performance of the communication module (which influences the overhead values) using the NS2 network simulator. The transmission range was set to 25m using the 2-ray ground propagation model and the 802.11b MAC layer was used. Though the range of 802.11b can be higher than 25m, higher ranges require more power, which is not desirable on power constrained mobile devices. Host movement was modeled using the random waypoint mobility model with hosts moving at a uniform speed of 1.7 m/s, which is close to human walking speed.

With mobile hosts, it is not appropriate to compare performance as a function of the number of hosts solely as additional factors are involved such as the speed of the hosts and the total area that a group of hosts are responsible for. Hence, we use a concept called *upper bound coverage* to determine the fraction of the total area that is within communication range of at least one host in a single second. The formula for coverage is $(h/a)(\pi.r^2 + 2.s.r)$ where h is the number of hosts, a the total area, r the communication radius of hosts, and s the speed of the hosts. The second term gives the instantaneous area that falls within the communication radius of a single host plus the differential area covered in the second under consideration. This is multiplied by the number of hosts to give the upper bound covered by all hosts and then divided by the area to give a fraction in the range $0 \leq \text{coverage} \leq 1$ with 1 indicating full coverage and 0 indicating no coverage. The coverage upper bound is reached only if every point in the area is covered exclusively by one host. In practice, the coverage is lower than the upper bound due to certain points falling within the range of more than one host. Holding area a constant, increasing the number of hosts, speed or the communication radius influences coverage positively while increasing area holding the other quantities constant influences the coverage negatively. Intuitively, more coverage means that it is more likely for a host to be at a particular location whereas less coverage indicates that a host is less likely to be at a specific location. We used this environment to simulate the execution of randomly generated workflows, the results of which appear below. Each data point is an average of 30 runs.

Expt. 1 - Completing Workflows. In this experiment, we examined the influence of our protocols on workflow completion. As a baseline, we used a protocol that delivers data and subscriptions directly without using intermediate hosts, i.e., in a peer to peer manner during opportunistic encounters. We measured (1) the number of tasks completed and (2) the number of tasks that failed due to a communication error when using the baseline protocol as well as each of our three schemes. The remaining tasks failed due to a dependency on the tasks that failed due to communication errors. The results are shown in Figure 5. Each of our schemes outperformed the baseline with Scheme 3 showing the best performance. All schemes showed close to 100% completions when the coverage was greater than 0.25. This illustrates that workflows are more likely to complete when one of our schemes is used. In the case of the workflows that failed to complete using our scheme, the reason was almost always due to aberrant mobility patterns where a host isolated itself from the rest of the network. It should be noted that we set an upper bound of 25000 seconds for each trial. This upper bound is 200% of the worst case time in which a workflow was actually completed.

Expt. 2 - Influence of Coverage on Overhead. Figure 6 shows the relation between coverage and overhead. Each data point is an average of executing 50 workflows. As can be seen, an increased coverage of the area in which the workflow is executing leads to lower overhead, primarily due to the availability of more routing options. An interesting observation is that there was a lot of variance in the data points for lower values of coverage. This can be explained as follows: the coverage captures the area that a host "touches" over the interval of a second averaged over all hosts participating in the workflow. When low coverage is prevalent, hosts may cover a the "correct" subset of the total area in which a large fraction of the workflow tasks must take place. This can result in low overhead. However, if the hosts cover a different subset of the area that does not include many tasks in the workflow, the overhead increases due to non-availability of hosts to perform tasks or route results. The notion of "correct coverage" is inherently tied to the

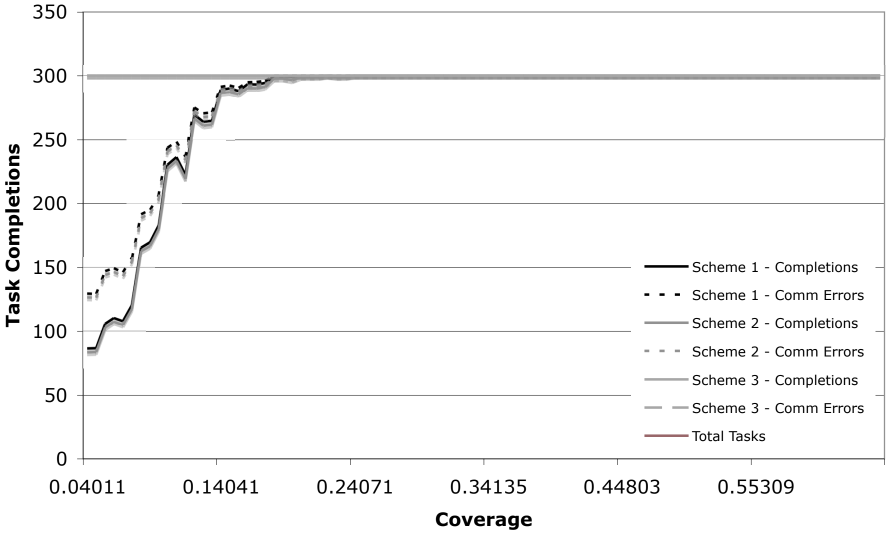

Fig. 5. Completions w.r.t. coverage

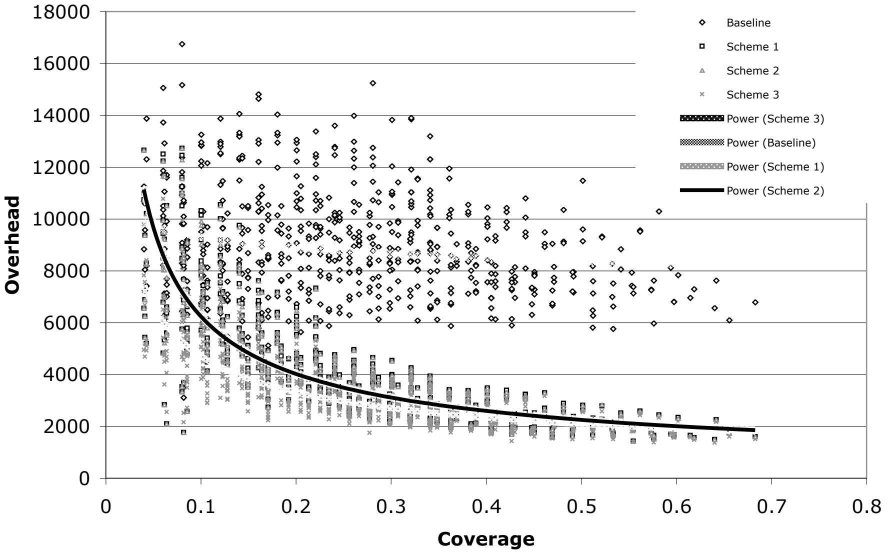

Fig. 6. Overhead w.r.t. of coverage

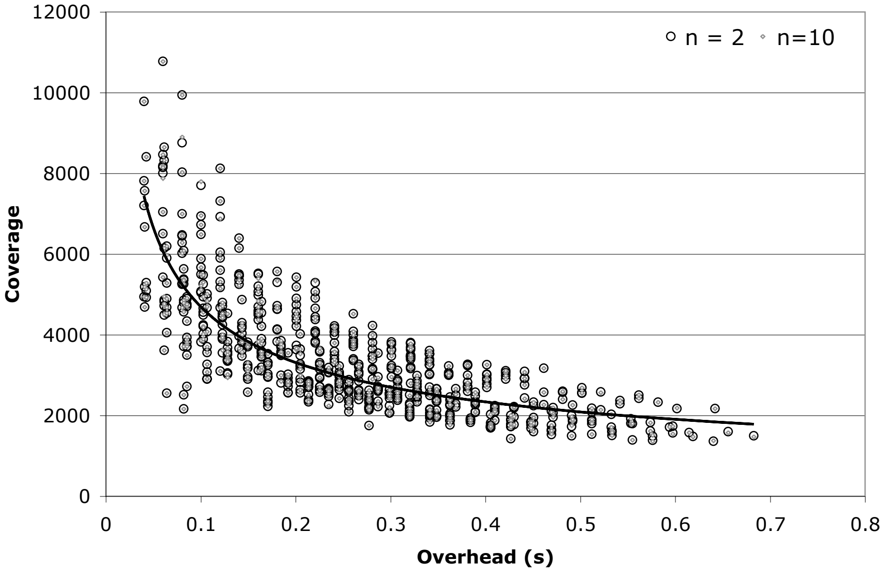

Fig. 7. Varying the value of 'n' in Scheme 3

workflow being executed (different workflows need different subsets of the area covered). Hence, coverage gives a sense of the performance but may be subject to high variances at the bottom of the scale.

Expt. 3 - Influence of 'n' on Scheme 3. Our final experiment involves a deeper study of Scheme 3 of our routing protocol. In Scheme 3, data and subscriptions are allowed to be routed outside their permitted range for a limited number of hops, i.e., the value of 'n'. In this experiment, we show the overhead of Scheme 3 with two values of 'n' - low and high. As can be seen in Figure 7, the value of 'n' did not significantly improve performance. The higher value of 'n' completed on average only a few seconds faster. We attribute this small difference to the fact that we chose environments where host density was not excessively sparse, and the fact that hosts encountered each other sufficiently often to pass on messages. We do expect to see a bigger difference for extremely low values of coverage.

Our results indicate that our routing protocol based on the task numbers improves upon the performance of naive approaches in terms of workflow completions as well as the overhead associated with communication. In addition, due to only a limited flooding of packets (within the range of task numbers), the packets in the network are significantly lower, leading to reduced bandwidth and power usage. These results are encouraging. However, we do intend to refine our approach to achieve more efficiency in future work.

5 Related Work

A WfMS is the piece of software that executes a compatible workflow specification. Today, innumerable WfMSs are available as both commercial and open source software such as FLOWer [3], AgentWork [20], Caramba [8], Groove [6], and I-Flow [15]. ActiveBPEL [9], JBoss [19], Oracle Workflow Engine [18] are just a few of the engines available today that run BPEL workflows while BizTalk [7] supports XLANG. Each of these engines is designed for orchestrated operation in wired settings.

In [5], message passing is used to distribute data in a wired setting while MoCA [14] uses proxies for distributed control. MoCA has some design features that support mobile environments while Exotica/FDMC [2] describes a scheme to handle disconnected mobile hosts. In AWA/PDA [26], the authors adopt a mobile agent based approach based on the GRASSHOPPER agent system. WORKPAD [11] is designed to meet the challenges of collaboration in a peer-to-peer MANET involving multiple human users. WORKPAD's shortcoming is that it requires at least one member of a MANET to be connected with a central entity that coordinates the mobile devices. Our work is targeted to an environment similar to that of WORKPAD. However, our approach is different in that we use choreography rather than a central coordinator.

With a choreography-based system, a leading concern is the process by which a workflow is distributed across various participants and then executed. In [21], the authors describe the process by which a monolithic workflow specification can be fragmented and eventually distributed across multiple hosts while in [5], the authors parse a BPEL specification, discard all the structural constructs and use the *link* construct to build a more graph-like specification. Several systems exist that achieve partial choreography, a survey of which appears in [16]. OSIRIS [24] is one such system where individual nodes maintain a hyperdatabase (HDB) to which is pushed service execution requests by a set of global process repositories. The choice of who to push the request to is handled by established load balancing techniques. $\text{ADEPT}_{Distribution}$ [4] describes a scheme for distributed execution of workflows such that the number of network messages is minimized. Additional efforts are ongoing to define protocols and standards for choreography such as in WS-CDL [1].

In summary, there are large bodies of work in orchestrated systems and languages supporting orchestrated systems in wired settings or environments with limited mobility. Our work advanced the state of the art by bringing workflows to the most dynamic type of mobile networks - MANETs - via the design of a lightweight, decentralized, and choreographed WfMS.

6 Conclusion

WfMSs that provide orchestrated workflow management across stable wired networks are a proven technology today. However, when a WfMS is developed with a mobile environment in mind, the centralized nature of orchestrated systems must give way to distributed and choreographed systems. In this paper, we described CiAN, a WfMS designed for MANETs that uses choreography of services to complete workflow tasks. CiAN uses a publish-subscribe-like protocol that takes results from a task and delivers them to the host responsible for executing the immediately succeeding tasks without going through a central coordinating entity. This protocol was developed with MANETs in mind where routes between hosts are transient and can break in an unpredictable manner. In our evaluations, we found that the calls to the services that occur locally on individual hosts took significantly less time than the process of communicating data and results between hosts. We evaluated three variants of our communication protocol all of which showed 100% completion when coverage upper bound was greater than a quarter of the total area and with reasonable amounts of overhead relative to the total specified duration of the workflow. We plan to build on this work and add new features like workflow cycles and error management in future work.

References

1. WS-CDL v1.0 (November 2005), `http://www.w3.org/TR/ws-cdl-10/`
2. Alonso, G., Gunthor, R., Kamath, M., Agrawal, D., El Abbadi, A., Mohan, C.: Exotica/FDMC: A Workflow Management System for Mobile and Disconnected Clients. Parallel and Distributed Databases 4(3) (1996)
3. Athena, P.: Flower User Manual (2001)
4. Bauer, T., Dadam, P.: A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration. In: Proc. of CoopIS, pp. 99–108 (1997)
5. Chafle, G., Chandra, S., Mann, V., Nanda, M.G.: Decentralized Orchestration of Composite Web Services. In: Proc. of the 13th Intl. WWW Conference, pp. 134–143 (2004)
6. Microsoft Corp. Groove Virtual Office, `http://www.groove.net/home/index.cfm`
7. Microsoft Corp. The BizTalk Server, `http://www.microsoft.com/biztalk/`
8. Dustdar, S.: Caramba - A Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Teams. Distributed and Parallel Databases 15, 45–66 (2004)
9. Active Endpoints. ActiveBPEL engine, `http://www.active-endpoints.com/active-bpel-engine-overview.htm`
10. Hackmann, G., et al.: Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 503–508. Springer, Heidelberg (2006)
11. Mecella, M., et al.: WORKPAD: an Adaptive Peer-to-Peer Software Infrastructure for Supporting Collaborative Work of Human Operators in Emergency/Disaster Scenarios. In: Proc. of CTS (May 2006)
12. Sen., R., et al.: Knowledge Driven Interactions with Services Across Ad Hoc Networks. In: Proc. of ICSOC, pp. 222–231 (2004)
13. Sen, R., et al.: Coordinating Workflow Allocation & Execution in Mobile Environments. In: Murphy, A.L., Vitek, J. (eds.) COORDINATION 2007. LNCS, vol. 4467, pp. 249–267. Springer, Heidelberg (2007)
14. Sacramento, V., et al.: An Architecture supporting the development of Collaborative Applications for Mobile Users. In: Proc. of WETICE 2004, pp. 109–114 (2004)
15. Fujitsu. i-Flow Developers Guide (1999)
16. Hahn, C.: A Comprehensive Investigation of Distribution in the Context of Workflow Management. In: Proceedings of ICPADS, pp. 187–192 (2001)

17. Haustein, J., Siegel, J.: ksoap (2006), `http://www.ksoap.org`
18. Oracle Inc. Oracle Workflow, `http://www.oracle.com/technology/products/integration/workflow/workflow_fov.html`
19. JBoss Labs. JBoss Application Server, `http://www.jboss.com/docs/index`
20. Muller, R., Greiner, U., Rahm, E.: AgentWork: A Workflow System Supporting Rule-based Workflow Adaptation. Data and Knowledge Engineering (2004)
21. Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A.K., Weikum, G.: From Centralized Workflow Specification to Distributed Workflow Execution. Journal of Intelligent Information Systems 19(2), 159–184 (1998)
22. Perkins, C.E., Royer, E.M.: Ad hoc On-Demand Distance Vector Routing. In: Proc. of WMCSA, pp. 90–100 (1999)
23. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs (1981)
24. Schuler, C., Weber, R., Schuldt, H., Schek, H.-J.: Scalable Peer-to-Peer Process Management The OSIRIS Approach. In: Proc. of ICWS, pp. 26–34 (2004)
25. Sen, R., Roman, G.-C., Gill, C.: Distributed Allocation of Workflow Tasks in MANETs. Technical report, Washington University in St. Louis (2007)
26. Stormer, H., Knorr, K.: PDA- and Agent-based Execution of Workflow Tasks. In: Proceedings of Informatik 2001, pp. 968–973 (2001)
27. van der Aalst, W.M.P.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)