Network Awareness and Mobile Agent Systems

Wilmer Caripe, George Cybenko, Katsuhiro Moizumi, and Robert Gray
Dartmouth College

ABSTRACT Most current computer applications are insensitive to changing network conditions. With the growing demand for wireless, satellite, and other highly volatile computer communications networks, however, applications that are robust in the presence of network volatility must be designed and built. Network-robust applications are of great interest in military situations today, and we expect that interest to grow in industrial and eventually consumer environments as well. Mobile agents are one way to realize such applications, especially when used in a wireless environment. This article discusses issues and results related to the problem of making computer applications network-aware and reactive to changing network conditions. It contains a short overview of our work on mobile agents as well as a tutorial on network sensing from the agent perspective. Some prototypes of network-sensing systems and network-aware mobile-agent applications are presented.

"Classical" networking has effectively separated network and communications issues from end-user applications through the abstractions of the open systems interconnection (OSI) framework and other reference models. These models have successfully defined protocols by which developers could focus their work at a level appropriate to their development needs — physical, data link, medium access, network, transport, and applications layers, for example. This architectural concept has been enormously successful and implemented almost universally. However, several capabilities have fallen through the cracks of these layers. Once an application establishes a session with a remote host application, the practice in transport-layer protocols like TCP has been to "do one's best" to keep that connection viable with essentially no fallback options for applications at either end. In cases like this, if a link fails, the applications that had active sessions running over that link enter some sort of error-handling mode and effectively terminate. This point of view makes sense when such failures are truly exceptions. However, if such failures occur relatively frequently, and if distributed applications must operate in networking infrastructures that do not offer adequate quality-of-service mechanisms, new ideas are needed.

Network awareness, the property of having knowledge about the current status of underlying network resources, represents a new approach to this issue [1–5]. Distributed network-aware applications, executing in volatile network environments, have the ability to react in response to changes in the status of the network, with the ultimate goal of minimizing the impact of these changes on the application's performance.

Mobile agents are software systems designed precisely to handle volatile network environments, moving from machine to machine while preserving their state information. Network communication with other applications is temporarily suspended and resumed once a stable host has been reached. This allows critical applications to survive network link failures and degradations. Mobility also allows for more effective use of bandwidth in situations where processing should be done remotely at the data location instead of locally after downloading a large dataset. In order to make mobile agents respond to such challenges, the underlying mobile-agent system should provide a network-awareness infrastructure. Only then can the agents make decisions about how to continue their operation most effectively.

This article discusses the need for network-awareness in mobile-agent systems and proposes mechanisms for providing the necessary infrastructure to accomplish that goal. In doing so, we explore the potential for using the mobile agents themselves and other communicated objects as carriers of network status information within such an infrastructure.

The reader should be aware that these are relatively new areas of investigation with much work remaining to be done. Few robust mobile-agent systems presently exist. Fewer still are publicly available. Ideas and results for network-aware mobile-agent systems are only now beginning to surface, and this article describes our current work in this area.

The second section presents a motivation for network awareness in mobile agent systems. The third section discusses issues surrounding network sensing for and by mobile agents, examining the tradeoffs inherent in various approaches. The fourth section describes our initial implementations and experiments with network-aware mobile agents. The fifth section presents related work. The last section is the conclusion.

Motivation for Network Awareness in Mobile Agent Systems

To establish the need for network awareness in a mobile-agent system, we focus our attention on active hybrid networks (a combination of reliable terrestrial links, wireless links and both fixed and volatile mobile nodes). Mobile agents have great potential and important applications in this kind of environment. Such networks present a major challenge for any distributed application: network topology can change constantly, bandwidth can be limited, latency can be high and communication links can be down because of natural causes, detection concerns or power conservation. Hence, mobile agents executing in hybrid networks need to be aware of relevant changes in the availability of network resources (*awareness*), and need to react accordingly to guarantee successful performance of their tasks (*agility*) [1]. The following sample problems illustrate situations in which these two properties are appropriate for mobile-agent systems.

A Mission-Critical Agent — A mobile agent providing a high-availability service on a computer connected to a wireless network should be able to detect events such as congested or noisy communication links and high processing loads. The mis-

Supported by Air Force Office of Scientific Research grant F49620-97-1-0382.

 0163-6804/98/$10.00 © 1998 IEEE

sion-critical agent should then move to another computer with a safer profile to guarantee uninterrupted service availability. Directories or proxy servers are used to ensure location transparency for the service provided by the mobile agent. Related work has shown that network-aware placement of distributed applications across the network can provide performance gains compared to network-oblivious placement [4].

The Traveling Agent Problem — While performing a task that requires visiting a sequence of remote servers, a mobile agent should be able to determine the most efficient and safest route based on current network conditions. For example, the agent can get stranded at a node with a poor wireless link. In this case, network awareness is instrumental for the mobile agent to plan its visit in a way that avoids noisy or congested links and that favors hosts with the lowest processing load. Although this planning process is formulated as a dynamic resource allocation problem that cannot be efficiently solved using traditional optimization methods due to space and processing-time limitations, some short-term decisions can be made based on available network information [6].

In order to realize the benefits of network-aware mobile agents in such sample problems, an appropriate infrastructure is needed to provide estimates about the status of the computer network. Mechanisms for providing network awareness to traditional distributed applications are a reasonable starting point for developing such an infrastructure in a mobile-agent system, as will be shown in the next section.

At the same time, it is important to note that previous work on network-adaptive services such as available bit rate (ABR) essentially focuses on point-to-point network status (or multipoint-to-point status, as in the case of multicast ABR) [7]. The above two mobile-agent scenarios demonstrate the need for more comprehensive network information that cannot be inferred from local point-to-point link performance information. For instance, a mission-critical agent must be able to migrate to a new location that has adequate communication and computation performance. This requires knowledge of the current environment as well as the candidate remote environments. Similarly, a traveling agent must plan a route based on remote point-to-point statistics.

Another important issue is whether agent policies for allocating distributed network resources are stable and effective. Work on such cooperative strategies is only now beginning in a new DARPA-funded project, "Resource control in large-scale mobile-agent systems."[1]

Different Approaches to Network Awareness for Distributed Applications

Network sensing or *monitoring* is the process of collecting information about network performance. An important process that relies on this information is *prediction* of future network performance. While prediction is an important topic, it presents innumerable open issues that are beyond the scope of this article. The rest of this section provides a survey of the different approaches to network sensing.

Existing network-aware systems monitor such parameters as latency, available bandwidth, packet loss rate, and CPU load for different computers in the network [1, 3, 8]. Here we will use the term *network monitor* to refer to an entity in charge of the network-sensing tasks in a computer or network. A network monitor is usually a software entity.

The network-sensing process can be classified according to the criteria in Table 1. One classification is based on the amount of traffic generated during the monitoring process: in passive monitoring, network monitors piggyback status information on existing messages, whereas in active monitoring, network measurements are done by sending additional control messages [4].

Network monitoring can also be classified according to whether it is performed on demand or continuously. On-demand monitoring occurs when applications ask the monitor to collect status information about a certain resource in an online fashion. In continuous monitoring, on the other hand, the monitor informs the application when the status of a previously requested resource changes in a certain way (e.g., falls below a predefined threshold). The latter scheme requires mechanisms for applications to register their resource interests with the monitor, either synchronously or asynchronously, as discussed in [3].

Finally, depending on how status information is replicated, network monitoring can be centralized or distributed. In the centralized case, status information from the whole network is consolidated at a central host and shared by all other hosts (most commonly, this information is mirrored at several central hosts). In the distributed case, monitors collect only local network status information and obtain non-local status information on demand from network monitors on other computers. The first scheme is not scalable, since the network monitors would maintain virtually the same status information, leading to a large amount of wasted storage. In the second scheme, collaboration between monitors is necessary if applications need status information about resources in computers outside the vicinity of the local network monitor. An example of the latter is presented in [1], where UDP messages are used by collaborative resource-monitoring daemons in different computers.

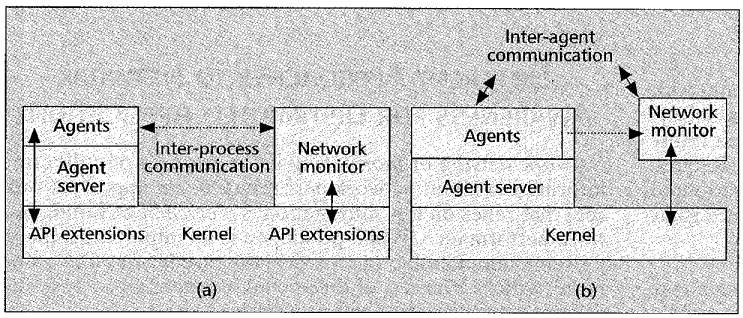
Criteria	Type of monitoring	
Traffic generated for monitoring purposes	Active	Passive
	Generates extra control messages to collect status information	Piggybacks control information on packets traversing the network
Monitoring frequency	On demand	Continuous
	Collects resource information only when requested by application	Informs registered applications about changes in resource status
Replication of information	Centralized	Distributed
	Collects and stores status information for all resources on a central machine	Collects and stores information about local resources only, exchanges information with monitors in other computers on demand

■ **Table 1.** *Taxonomy of network monitoring.*

Network-Aware Mobile Agent Systems

Based on the review given in the previous section, we present two approaches that can be used for providing network awareness to mobile-agent systems. Network-aware mobile agents have the capability to request resource status information from a network monitor.

[1] *In response to DARPA RFP BAA 98-01.*



■ **Figure 1.** *Mechanisms for network monitoring in mobile agent systems.*

All the ideas presented here are devised for a mobile-agent system with similar capabilities to those found in D'Agents, a mobile agent system developed at Dartmouth College. An overview of this system will be given in the next section. D'Agents and the two approaches presented in this subsection represent the framework for our ongoing and future work in network-aware mobile-agent systems.

Figure 1a shows an autonomous network monitor, which is a separate entity providing resource-monitoring services to all the applications in a host, not just to mobile agents. Under this scheme, mobile agents trying to obtain information from the network monitor do so through APIs, modified operating system calls [3], or inter-process communication mechanisms [1]. The network monitor is an independent entity, which can be implemented under any of the categories described in Table 1. Sumatra, presented in [1, 13], is a network-aware system that supports migration and uses a similar network-sensing model.

Figure 1b shows an approach where a mobile-agent system includes its own private monitoring agent. As in the previous case, the monitor interacts with the operating system to obtain local network information. If the monitor requires data about non-local resources, however, it requests the data on demand from another monitor by leveraging those sessions that are established between mobile-agent servers when an agent migrates from one machine to another. Note that the effectiveness of this *passive* collaborating strategy will depend on the *assumption* of locality — that agents will most likely want to learn resource status for those computers with which the local agent server maintains regular interaction. If this assumption does not hold, a mechanism that combines the piggybacking scheme with explicit requests can be devised. For example, if the local monitor is asked about resource status of another computer, the monitor waits until the local agent server establishes a *migration* session with the agent server at the targeted computer. If this does not occur within a given period of time, the monitor instead sends a dedicated agent to the remote machine to collect the information from the remote monitor.

Since the monitor in Fig. 1b is implemented as an agent itself, other agents trying to submit resource status requests to the monitor can do so by using available inter-agent communication. If the system were implemented using D'Agents, for example, agents would use a remote procedure call (RPC)-like mechanism that allows communication between agents on the same machine or on different machines [9].

There might be cases where aggregation methods are applied to performance measurement, and thus there is only one monitor collecting information that will be representative for all the computers in a certain zone (e.g., a switched LAN or a high-availability computer cluster). For these situations, those computers not designated as the *zone* monitor will have a *proxy* monitor instead of a full-featured network monitor. The proxy monitor at each computer will be in charge of rerouting the requests to the main network monitor in the *zone*, which will then perform the required status collection tasks and return the information back to the proxy. This will in turn redirect the query results to the appropriate application.

Our current work, which is explained in detail later, is mainly based on the scheme of Figure 1a. Our ultimate goal is to implement a completely integrated system following the architecture of Figure 1b. To the best of our knowledge, no implementation of this model has been done. Many issues remain open regarding the latter scheme, some of which are discussed later.

Mobile Agents for Network Monitoring

Ideally, distributed network monitors should be capable of sharing status information without generating significant additional traffic overhead, especially when the underlying network suffers bandwidth limitations (as is the case for wireless networks in general).

Based on the concept of passive monitoring mentioned before, and the idea of *active networks* presented in [10], we believe that mobile agents are an appropriate way to enable collaboration among monitors.

Each agent can carry network status information along with its own logic and data. At each node they visit, agents interact with the local network monitor through the agent server: visiting agents obtain from the local monitor network information that is more recent than their own or complementary to it. At the same time, the local monitor updates its databases with whatever new status information visiting agents bring from the rest of the network. Hence, no extra messages are generated by the monitors' collaborating activities.

Moreover, by understanding the network information they carry, agents can locally optimize the distribution of network status information in situations involving limited bandwidth or high network congestion by, for example, bifurcating themselves to carry network information or data at different levels of granularity. Agents carry out this self-replication process in response to communication delays estimated along their path, allowing more important information to be given higher priority.

Of course, there are many open issues regarding the process described in the previous paragraphs, and research in this area is only beginning. A consistent way to represent network information has to be devised in order to allow agents to analyze it and exchange it for prioritization purposes. Also, robust self-replication mechanisms have to be defined, taking into account the security and consistency issues involved in assembling and disassembling agent network-status contents. Moreover, hierarchical and clustering approaches might be necessary to logically organize large networks, therefore allowing for scalability and complying with the low-overhead premise previously stated. More information about managing large networks at different levels of granularity can be found at the Proactive Problem Avoidance Project's Web site at http://www.cs.rpi.edu/~kaploww/finalproposal.html.

Our Current Work on Network Monitoring and Agents

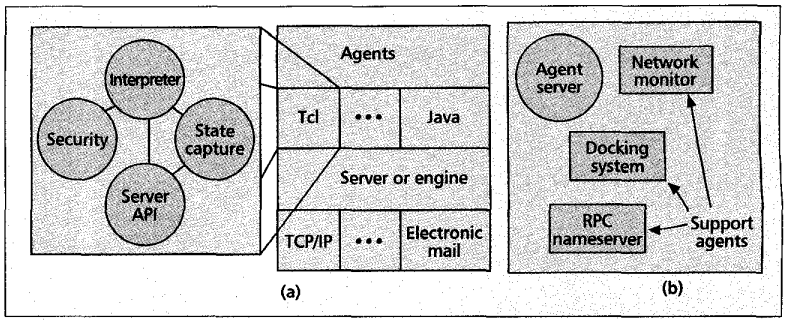
An Overview of D'Agents

D'Agents[2] is a mobile-agent system whose agents can be written in Tcl, Java, Scheme, and Python. D'Agents is in active use at numerous academic and industrial research laboratories, including those at Lockheed Martin, Siemens, and the University

[2] *D'Agents was once called Agent Tcl.*

of Bordeaux. It is starting to find its way into production-quality applications. The current public release supports Tcl agents only. It provides migration, low-level communication, and significant security mechanisms. New versions, with support for all four languages, will be publicly released in spring and summer 1998.[3]

Like all mobile-agent systems, the main component of D'Agents is a server that runs on each machine. When an agent wants to migrate to a new machine, it calls a single function, *agent_jump*, which automatically captures the complete state of the agent and sends this state information to the server on the destination machine. The destination server starts up an appropriate execution environment (e.g., a Tcl interpreter for an agent written in Tcl), loads the state information into this execution environment, and restarts the agent from the exact point at which it left off. Now the agent is on the destination machine and can interact with that machine's resources without any further network communication. In addition to reducing migration to a single instruction, D'Agents has a simple layered architecture that supports multiple languages and transport mechanisms. Adding a new language or transport mechanism is straightforward: the interpreter for the new language must support two state-capture routines, and the "driver" for the new transport mechanism must support asynchronous I/O and a specific interface. As indicated above, the primary language is Tcl, with support for Java and Python nearly complete, and support for Scheme in progress. The primary transport mechanism is TCP/IP.



■ **Figure 2.** *a) Architecture of the D'Agents system; b) support agents at each machine.*

Figure 2 shows the architecture of D'Agents. The core system, which appears on the left, has four levels. The lowest level is an interface to each available transport mechanism. The next level is the server that runs on each machine. This server has several tasks. It keeps track of the agents running on its machine, provides the low-level inter-agent communication facilities (message passing and binary streams), receives and authenticates agents arriving from another host, and restarts an authenticated agent in an appropriate execution environment. The third level of the architecture consists of the execution environments, one for each supported agent language. All current D'Agents languages are interpreted, so our "execution environments" are just interpreters, namely a Tcl interpreter, a Scheme interpreter, a Python interpreter, and the Java virtual machine.

The last level of the architecture are the agents themselves, which execute in the interpreters and use the facilities provided by the server to migrate from machine to machine and to communicate with other agents. Agents include both moving agents, which visit different machines to access needed resources, as well as stationary agents, which stay on a single machine and provide a specific service to either the user or other agents. The agent servers provide low-level functionality. Dedicated service agents provide all other services at the agent level, as Fig. 2b shows. Such services include navigation, high-level communication protocols, and resource management.

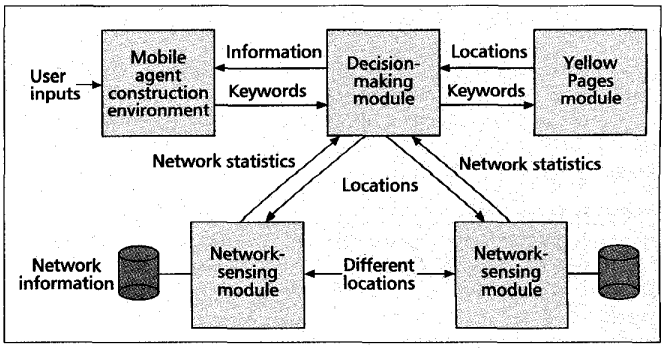
D'Agents includes an important network-aware feature: a docking system that lets agents transparently jump off an intermittently connected computer (such as a mobile laptop) and return later, even if the computer is connected only briefly to the network. Each mobile computer using D'Agents is associated with a dock computer. Using this scheme, when a mobile agent is unable to migrate to a target laptop (TL), it is automatically queued at the TL's dock machine until the TL reconnects to the network. The TL uses the network tool *ping* to poll its dock computer at regular intervals. Since a dock computer is by definition a machine permanently connected to the network, a successful poll to this computer implies available connectivity to the network. At this point, the TL notifies its dock of its new network address. The dock computer then forwards all waiting agents to the TL and receives all the agents queued at the TL waiting to migrate to some other computer [9]. The current docking system is rather simple; ongoing work aims to make it more efficient and robust (e.g., what if the dock machine is down?).

The Agent Migration Planning Process

Suppose you are shopping for a specific item known to be sold in $n + 1$ stores. The probability, p_i, that store i has the item is known and independent for different stores. Moreover, it takes a known time, t_i, to navigate through store i to the section where the item is stocked, thereby determining whether the item is available or not. Going from store i to store j requires travel time l_{ij}. Given that information, and starting and ending at store 0, what is the minimal expected time to find the item or conclude that it is not available?

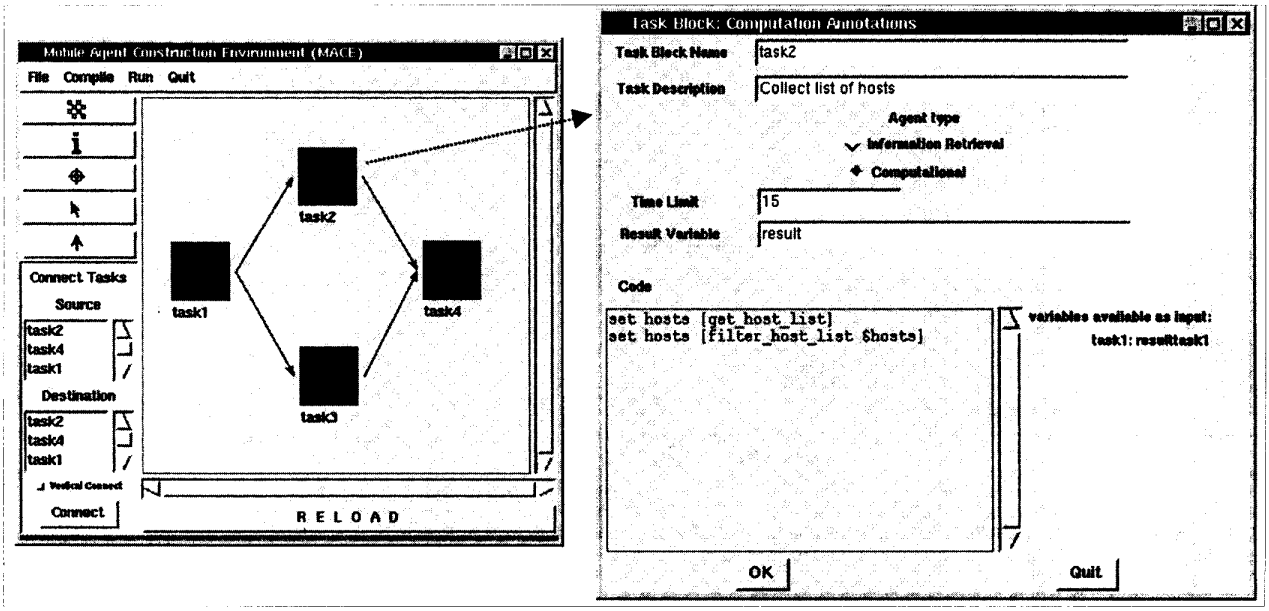
The shopping analogy describes the essence of the agent migration-planning problem. In a mobile-agent context, the stores are information servers such as databases or Web servers. The probabilities of success can be estimated from relevance scores given by search engines. Compute times and latencies might be obtained from network status monitors and directory services [6].

We have implemented a simple prototype to illustrate this planning process. It follows the scheme depicted in Fig. 3. In this approach, a GUI interface — the Mobile Agent Construction Environment (MACE) for D'Agents [11] — allows the user to graphically specify the information retrieval tasks



■ **Figure 3.** *The agent planning process.*

[3] *See http://www.cs.dartmouth.edu/~agent for software, documentation, and related papers.*



■ **Figure 4.** *An example of the MACE interface.*

as a workflow. The user also specifies the keywords that will drive the information retrieval process. An example of the MACE interface is given in Fig. 4. In Fig. 4a the main screen shows four user-defined tasks. In Fig. 4b the snapshot shows how the user specifies the computation to be performed by *task2*, in this case using the Tcl programming language. Once the user has constructed the agent with MACE, she launches the agent into the network.

The agent's decision-making module then consults the Yellow Pages module, which is implemented as a separate mobile agent, to identify host locations where information related to the specified keywords might be found and the probability of finding relevant information at each of those locations.

The decision-making module executes a two-step look-ahead exhaustive search to determine which computer should be visited and searched next. As part of this process, an agent is sent to each computer in the location list provided by the Yellow Pages. Each agent consults the local network information at the corresponding target node with respect to the other nodes in the list. Based on that data, each agent determines which location would be the best choice to visit next from the target node if the user-specified task were being executed there. This information, along with the corresponding network statistics, is sent back to the original node so that the decision-making module can finish its calculation. Once all the agents have come back with relevant information, a decision is made concerning to which computer the search should go next. The decision process is repeated at every node that the working agent visits while performing its task. While the agent is performing its information retrieval subtask at each node it visits, its decision-making module in parallel sends out the network-sensing agents.

In special cases, we have discovered efficient optimal algorithms for ordering the sites to be visited. A sample result from [6] is as follows.

THEOREM — If the latencies are all constant so that $l_{ij} = l$, then the optimal (minimal expected time) ordering is obtained by visiting the sites in decreasing order of p_i/t_i. Moreover, if sites are organized into N subnetworks so that latencies within subnetworks are constant and latencies across subnetworks are also constant but different from intra-subnet latencies, then the optimal route can be found in no more than $C(n_1 + 1)(n_2 + 1)...(n_N + 1)$ steps where the kth subnetwork has n_k sites.

Note that this reduces to exponential complexity in the case where each subnetwork has one site.

Network Sensing Module — This module is implemented as a standalone service that collects network information on a periodic basis on each node. The module keeps up-to-date information stored in a local database, which can be accessed by mobile agents through an API implemented as extended Tcl commands. Currently, the module collects only latency and bandwidth information on each node with respect to all the other computers specified in a master file[4] (host set). These network statistics are collected using a mechanism similar to the one used by the *ping* program. Instead of using Internet Control Messaging Protocol (ICMP) messages, a customized server is set up on each computer to accept ECHO-like UDP messages and retransmit them to the original sender, which in turn calculates the round-trip time and henceforth the estimated latency between the two nodes. Available bandwidth is calculated by measuring difference in round-trip time for probe packets of different sizes.

The monitor stores data about the network status between two computers in a UNIX *dbm* file. This database is indexed by a combination of the addresses of the two computers. Thus, using the API mentioned above, Tcl agents can open a session to access the database, retrieve latency and bandwidth information for any two given nodes, and close the session. If an agent attempts to retrieve information through the API, but the data is not found in the local database, an indication is returned by the retrieving function so that the agent can proceed as described in the previous paragraphs to obtain remote network information.

All the collected network-sensing information can be considered a distributed database where each node maintains information about latency and bandwidth from that node to the rest of the hosts in the agent system. Practically, the over-

[4] *This file should be a superset of every set of locations provided by the Yellow Pages.*

[5] *In the current prototype this matrix is considered to be symmetric.*

all network information database can be considered a matrix where a generic entry (i, j) contains latency and bandwidth information from node i to node **j**.[5] In this scheme, each node maintains the corresponding row in the matrix. In the current prototype, the only mechanism for remote consultation (i.e., if node **z** wants to know the network information between nodes **i** and **j**), is to send an agent from **z** to either node and retrieve the data from there. Other collaboration schemes have been discussed, such as using the Simple Network Management Protocol (SNMP) for accessing the data distributed across the network, but none have been implemented yet.

Future Work

Our prototype shows the applicability of network-sensing mechanisms to mobile-agent systems. Key enhancements to this prototype are under study, including modifications to the scheme used to access remote network status information. The current prototype explicitly sends *messenger* agents to retrieve the data from other computers, generating extra traffic overhead. We will focus new enhancements toward the implementation of a network-sensing support agent that will be included in a future release of D'Agents.

Related Work

Some research has been done on adaptation of applications to changes in the computing environment, with a special focus on mobile computing. *Odyssey*, an experimental resource management system [3], provides operating system support for application-aware adaptation through extensions to the operating system APIs. The system monitors resource levels, notifies applications of relevant changes, and enforces resource-allocation decisions, while each application decides how to best adapt when notified. A programming-level abstraction is proposed in [2] to provide support for environment-aware applications. This guarantees portability of the application across multiple platforms and resource-management extensibility.

The Quality Assurance Language (QuAL) described in [12] provides language-level abstractions and a runtime system that allow applications to specify and negotiate communication and application-specific quality-of-service requirements. QuAL also provides feedback mechanisms for applications to adapt or re-negotiate quality-of-service demands when current quality-of-service requirements are not being met.

A network resource monitor operating in user space is presented in [5]. It was devised to provide network awareness to mobile applications so that they would exhibit more efficient network utilization. SPAND, a system that collects wide area network performance information by using passive measurements from a collection of hosts is presented in [8], which reviews benefits and challenges in using passive and cooperative measurements in network monitoring.

Concerning network-awareness in mobile-agent systems, only *Sumatra*, an extension of the Java programming language that supports mobile programs, and *Komodo*, a distributed network monitor — both presented in [1, 4] — provide support for mobile programs to adapt to changing network conditions. As a proof of concept, *adaptalk*, an Internet chat application, was developed. The *adaptalk* server takes advantage of adaptation and places itself appropriately in the network so as to minimize overall application response time. The results show that network-aware placement of distributed components across the network can provide performance gains compared to network-oblivious placement [4].

To the best of our knowledge, no work has been conducted on network-aware adaptation of mobile programs in a wireless network environment, although adaptation provides better performance in this kind of resource-limited environment.

Conclusion

We have described several ideas, approaches and implementations of network sensing for mobile agents. This is a new area of investigation and much work remains to be done both at the conceptual and experimentation levels. Techniques for effective, noninvasive network sensing have been proposed with some initial implementation experience. A major challenge is handling very large hybrid networks; methods for aggregating and abstracting total network status information into feasible sizes and usable forms remain an open problem.

References

[1] A. Acharya and M. Ranganathan, J. Saltz, "Sumatra: A Language for Resource-Aware Mobile Programs," *Mobile Object Systems*, J. Vitek and C. Tschudin, Eds., Springer Verlag, Apr. 1997, pp. 111–30.

[2] B. R. Badrinath and G. Welling, "Event Delivery Abstractions for Mobile Computing," Tech. rep. LCSR-TR-242, Dept. Comp. Sci., Rutgers Univ.

[3] B. Noble *et al.*, "Agile Application-Aware Adaptation for Mobility," *Proc. 16th ACM Symp. Op. Sys. Principles*, St. Malo, France, Oct. 1997.

[4] M. Ranganathan et al., Network-Aware Mobile Programs, *Proc. USENIX Annual Tech. Conf.*, Anaheim, CA, Jan. 1997.

[5] G. Welling and B. R. Badrinath, "Exporting Environment Awareness to Mobile Applications," Tech. rep. LCSR-TR-270, Dept. Comp. Sci., Rutgers Univ.

[6] K. Moizumi and G. Cybenko, "The Travelling Agent Problem," *Mathematics of Control, Signals and Systems*, Jan. 1998, available at ftp://witness.dartmouth.edu/pub/tap.ps.

[7] F. Bonomi and N. Giroux, "The Available Bit Rate Service," in "53 Bytes," *ATM Forum Newsletter*, vol. 3, issue 4, Oct. 1995.

[8] S. Seshan, M. Stemm, and R.H. Katz, "SPAND: Shared Passive Network Performance Discovery," *USENIX Symp. Internet Tech., and Sys,*, Dec. 1997, pp. 135–46.

[9] D. Kotz *et al.*, "Agent Tcl: Targeting the Needs of Mobile Computers," *IEEE Internet Comp.*, vol. 1, no. 4, July–Aug. 1997, pp. 58–67.

[10] D. L. Tennenhouse, and D. J. Wetherall, "Towards an Active Network Architecture," *Comp. Commun. Rev.*, vol. 26, no. 2, Apr. 1996.

[11] R. Sharma, "Mobile Agent Construction Environment," M.S. thesis, Thayer Sch. Eng., Dartmouth, 1997.

[12] P. Florissi and Y. Yemini, "Management of Application Quality of Service," Tech. rep. CUCS-022-94, Comp. Sci. Dept., Columbia Univ., 1994.

Biographies

WILMER CARIPE (wilmer.caripe@dartmouth.edu) finished his M.S. in computer engineering at Thayer School of Engineering, Dartmouth College, New Hampshire, in June 1998. He received his B.Sc. in computer science from Universidad Simòn Bolìvar, Venezuela in 1995. He is mainly interested in networking protocols, distributed network management, wireless networks, and network awareness in mobile agent systems. The focus of his M.S. thesis is on developing extensions for the Mobile IP protocol to improve performance of computers operating on multihop wireless networks.

GEORGE CYBENKO (george.cybenko@dartmouth.edu) is the Dorothy and Walter Gramm Professor of Engineering at Dartmouth College. Prior to joining Dartmouth, Cybenko was professor of electrical and computer engineering and professor of computer science at the University of Illinois at Urbana. At Illinois, he was also a director of the Center for Supercomputing Research and Development, which designed and built the CEDAR shared memory multiprocessor. He received his B.Sc. (1974) in mathematics from the University of Toronto and his Ph.D. (1978) in electrical engineering and computer science from Princeton. He has made significant contributions to theory and algorithms of signal processing, parallel computing, computer performance evaluation and neurocomputing. His current interests are in distributed information management, retrieval, and data mining problems.

KATSUHIRO MOIZUMI (katsuhiro.moizumi@dartmouth.edu) is a Ph.D. student in computer engineering at Dartmouth College. He received his B.E. and M.S. in electrical engineering from Waseda University, Tokyo, Japan, and received an M.S. in computer engineering from Clarkson University. His thesis work is on mobile agent planning problems.

ROBERT GRAY (rgray@dartmouth.edu) is an assistant research professor at the Thayer School of Engineering. He is the lead researcher and programmer for the core D'Agents system, the mobile-agent system discussed herein. He is primarily interested in the performance, security, and fault tolerance issues that arise in mobile-agent systems. He received his Ph.D. in computer science from Dartmouth College in 1997.