Mobile Networks and Applications 7, 163–174, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

Performance Analysis of Mobile Agents for Filtering Data Streams on Wireless Networks *

DAVID KOTZ **, GEORGE CYBENKO, ROBERT S. GRAY, GUOFEI JIANG and RONALD A. PETERSON
Dartmouth College, Hanover, NH 03755, USA

MARTIN O. HOFMANN, DARIA A. CHACÓN and KENNETH R. WHITEBREAD
Lockheed Martin Advanced Technology Laboratories, Camden, NJ 08102, USA

JAMES HENDLER
University of Maryland, College Park, MD 20742, USA

Abstract. Wireless networks are an ideal environment for mobile agents, since their mobility allows them to move across an unreliable link to reside on a wired host, next to or closer to the resources that they need to use. Furthermore, client-specific data transformations can be moved across the wireless link and run on a wired gateway server, reducing bandwidth demands. In this paper we examine the tradeoffs faced when deciding whether to use mobile agents in a data-filtering application where numerous wireless clients filter information from a large data stream arriving across the wired network. We develop an analytical model and use parameters from filtering experiments conducted during a US Navy Fleet Battle Experiment to explore the model's implications.

Keywords: mobile agent, mobile code, wireless network, performance analysis, information filtering, RPC

1. Introduction

Mobile agents are programs that can migrate from host to host in a network of computers, at times and to places of their own choosing. Unlike applets, both the code and the execution state (heap and stack) move with the agent; unlike processes in process-migration systems, mobile agents move when and where they choose. They are typically written in a language that can be interpreted, such as Java, Tcl, or Scheme, and thus, tend to be independent of the operating system and hardware architecture. Agent programmers typically structure their application so that the agents migrate to the host(s) where they can find the desired service, data, or resource, so that all interactions occur on the local host, rather than across the network. In some applications, a single mobile agent migrates sequentially from host to host; in others, an agent spawns one or more child agents to migrate independently.

A mobile-agent programmer, thus, has an option not available to the programmer of a traditional distributed application: to move the code to the data, rather than moving the data to the code. In many situations, moving the code may be faster, if the agent's state is smaller than the data that would be moved. Or, it may be more reliable, since the application is only vulnerable to network disconnection during the agent transfer, not during the interaction with the resource. For a survey of the potential of mobile agents, see [3,5].

These characteristics make mobile-agent technology especially appealing in wireless networks, which tend to have low bandwidth and low reliability. A user of a mobile computing device can launch a mobile agent, which jumps across the wireless connection into the wired Internet. Once there, it can safely roam among the sites that host mobile agents, interacting either with local resources or, when necessary, with resources on remote sites that are not willing to host mobile agents. Once it has completed its task, it can return to (or send a message to) its user, using the wireless network.

Clearly, the agent case avoids the transmission of unnecessary data, but does require the transmission of agent code from client to server. The total bandwidth consumed for code transmission depends on the agent size and arrival rate. For most reasonable agent code sizes and arrival rates, the savings in data transmission may be much larger than the code transmissions. Of course, each client's code could be preinstalled on the server.[1] This approach presupposes, however, that the clients are known in advance. In many of the environments that we consider, new clients with new code can appear at any time, and possibly disappear only a short while later. In scenarios like the one discussed in this paper, we need at least a dynamic-installation facility, and mobile agents give us the flexibility to move filtering code to any point in the net-

* An earlier version of this work appeared in the *Proceedings of the Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*. This research was supported by the DARPA CoABS Program (DARPA contracts F30602-98-2-0107 and F30602-98-C-0162 for Dartmouth and Lockheed Martin, respectively) and by the DoD MURI program (AFoSR contract F49620-97-1-03821 for both Dartmouth and Lockheed Martin).
** Corresponding author.

[1] In fact, most mobile-agent systems include, or plan to include, some kind of code-caching functionality, so that the agent code is transferred only the first time that an agent visits a machine.

work, and to move the code again as the situation changes. Although we do not consider such multi-machine scenarios in this initial paper, they will be an important part of future work.

In this paper we analyze the potential performance benefits of mobile agents in a typical data-filtering scenario. The scenario is based on a filtering experiment that was conducted during a US Navy Fleet Battle Experiment (FBE). In the FBE experiment, mobile agents were sent across a wireless link from the USS Coronado to a shore-based intelligence database, where they filtered incoming intelligence reports to find and return only those reports relevant to the Coronado's current mission. Although the scenario is drawn from an actual military exercise, it is sufficiently general to reflect many applications, from military applications in which soldiers monitor weather, terrain and troop movements, to commercial applications in which consumers monitor stock reports and news stories.

In our scenario there are numerous information producers, each of which pushes out a steady stream of information, such as weather observations, stock quotes, news stories, traffic reports, plane schedules, troop movements, and the like. Clearly each source has a different data rate and frequency. There are also numerous information consumers, whose computers are connected to a wireless network channel. We assume that the information streams gather at a gateway server, which then transmits the data across the wireless channel to the consumers. Although we model a single server machine, in a large system we expect that the server would be a multiprocessor or cluster, such as those used in large Internet servers today. Although we model a single wireless channel, the results are easily extensible to multiple channels, each with its own server, whether in separate or overlapping regions.

The overall picture is shown in figure 1.

Each consumer is interested in a different (but not necessarily disjoint) subset of the data. In particular, each consumer is interested in only a few information streams, and then only in some filtered set of items in those streams. For example, a traveler might monitor the weather stream, but not the stock stream, and of the weather stream, might care only about the weather in those locations that she will visit today. The first step requires no computation; the second may require some computation related to the size of the data stream. We model a consumer's interests as a set of *tasks*, all running on that consumer's single computer *client*.

We compare two approaches to solving this problem:

1. The server combines and broadcasts all the data streams over the wireless channel. Each client receives all of the data, and each task on each client machine filters through the appropriate streams to obtain the desired data.

2. Each task on each client machine sends one mobile agent to the server. These "proxy" agents filter the data streams on the server, sending only the relevant data as a message to the corresponding task on the client.

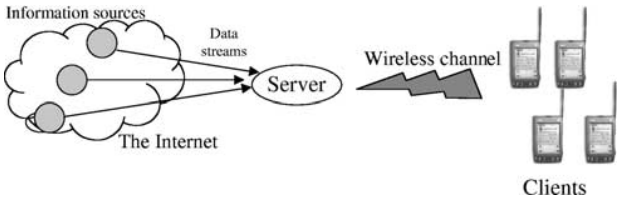

Figure 1. The scenario we analyze.

We use two performance metrics to compare these two techniques: the bandwidth required and the computation required. We can directly compare the usage of the two techniques, and we can evaluate the capacity needed in the server or the network. Clearly, the mobile agent approach trades server computation (and cost) for savings in network bandwidth and client computation, a valuable tradeoff if bandwidth is limited or if it is important to keep client weight and power requirements (and cost) low.

In the next section, we present the FBE experiment in more detail. Then, in two subsequent sections, we define the parameters that arise during the analysis, derive the basic equations, and interpret their significance. In section 4, we review the parameter values that we were able to obtain from the actual FBE experiment, and describe the experiments that we performed to obtain values for other key parameters. In section 5, we use these values to explore the performance space given by our model. We describe some related work in section 6 and summarize in section 7.

2. The Fleet Battle Experiment

In practice, the United States Navy (USN) Fleet Battle Experiment (FBE) series involves information-flow architectures that exemplify the general scenario described in the introduction. In FBE-Echo, the fifth in the series, Lockheed Martin Advanced Technology Laboratories (LM ATL) fielded CAST, a mobile-agent application that optimized the flow of critical information through bandwidth-limited, congested, and unreliable wireless networks [2]. As the results later in this paper indicate, the mobile-agent solution lowers bandwidth consumption in the limited experiment scenario and promises to be even more beneficial in a realistic, high-intensity operation.

The USN Maritime Battle Center in Newport, Rhode Island, conducts semi-annual FBEs in cooperation with the numbered USN fleets with the goal of streamlining and invigorating the Navy's warfare concept development, doctrine refinement and warfare innovation process. FBE-Echo was held in March 1999 in the San Francisco Bay area, and examined operational and tactical requirements for warfare in the years 2005–2010. More than 15 ships and 12,000 sailors and Marines from southern California participated. The FBE-Echo hypothesis was that "warfighting processes supported by new concepts and technology allow the Navy to enter and remain in the [coastal region] indefinitely with the

ability to provide protection, weapons fires and C4I[2] support to forces ashore."[3]

At FBE-Echo, CAST was integrated into the Full Dimension Protection Cell aboard the command ship, the USS Coronado. CAST spawned a set of mobile "scout" agents to correlate events indicative of impending Theater Ballistic Missile (TBM) launches and send filtered reports back to the USS Coronado. CAST then supplied alerts on TBM activity to the Air Operations Commander, who tasked fleet surveillance assets and launched a simulated preemptive strike. The scout agents traveled across the SIPRNET, the secure military Internet, from the USS Coronado to a simulated shore-based intelligence feed. The SIPRNET link between the USS Coronado and the shore was a wireless Super High Frequency (SHF) satellite communications link, whose bandwidth was 768 kbps.[4] Intelligence reports were approximately 150 bytes in size and arrived in bursts spaced about every four minutes, with each burst containing five to ten reports at a rate of one report every four seconds. The CAST scout agents were about 1 KByte in size.

The main benefit of CAST was that agents selectively scanned and filtered the incoming intelligence reports, forwarding only those that correlated to a significant event. The mobile scout agents carried selection specifications from the USS Coronado across the wireless link to the remote information sources, stayed there to monitor new information, and periodically sent back a much reduced set of data, saving both bandwidth and operator attention to irrelevant data. Each TBM event differs by the location, the initiating event, the sets of reports, and the timelines, so that an implementation without mobile filtering logic embodied in a mobile scout agent would have been cumbersome and complex.

Another benefit of CAST's mobile-agent approach was its ability to handle network outages. The satellite connection disconnected frequently, for a few seconds to an hour at a time. With a mobile-agent approach, the task of monitoring the database was uninterrupted once agents were resident on shore, although, of course, reports that passed the agents' filters had to be buffered until the link came back up. New agents were programmed to repeatedly attempt the ship-to-shore leap as long as the satellite connection was down.

In the exercise, the ratio of relevant to irrelevant reports was only about 0.5, since the simulated intelligence feed did not create a realistic number of extraneous "noise" events, and a scout agent was spawned approximately every 4 minutes. In an actual high-intensity operation, the number of extraneous reports will be much higher and drive the ratio of relevant to irrelevant reports down to at least 0.005. Especially in an urban scenario, a large number of reports would be generated by MTI (Moving Target Indicator) and ELINT (Electronic Intelligence) sources. The JSTARS MTI sensor, for example, is capable of reporting on 100 targets every second. A real conflict would involve several JSTARS and other sensor platforms. In such a situation, CAST would launch a larger number of scouts, up to a rate of 10 per second. The remaining parameters are identical for exercise and actual scenarios.

LM ATL has tailored the CAST mobile agents to several other military applications, including the DARPA Small Unit Operations program and US Army intelligence operations [8]. In each case, mobile agents proved to be effective in mitigating the effects of bandwidth-limited, unreliable, wireless networks.

To fully explore the general scenario presented by the FBE-Echo CAST experiments, we developed an analytic model. The model considers a more general scenario involving multiple clients (whereas CAST had one, the USS Coronado) and multiple independent streams of reports. In the rest of this paper we present our model and sample some of the performance space using specific parameters.

3. The model

Since the data is arriving constantly, we think of the system as a pipeline; see figure 2. We imagine that, during a time interval t, one chunk of data is accumulating in the incoming network buffers, another chunk is being processed on the server, another chunk is being transmitted across the wireless network, and another chunk is being processed by the clients. If the data arrives at an average rate of d bits per second, the average chunk size is td bits.

For the pipeline to be stable, then, each stage must be able to complete its processing of data chunks in less than t time, on average (figure 3). That is, $T_I \leqslant t$, $T_S \leqslant t$, $T_W \leqslant t$, and $T_C \leqslant t$. In the analysis that follows we work with these steady-state assumptions; as future work, we would like to explore the use of a queueing model to better understand the

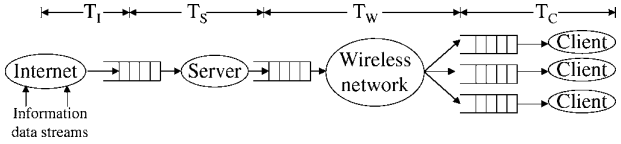

Figure 2. The scenario viewed as a pipeline.

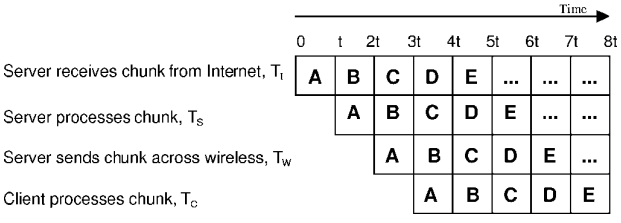

Figure 3. The pipeline timing diagram. The letters represent data chunks. For example, between time $3t$ and $4t$ chunk A is being processed by the clients, chunk B is being transmitted from the server to the clients, chunk C is being processed by the server, and chunk D is being received by the server.

[2] C4I is a military abbreviation for Command, Control, Communications, Computers and Intelligence.

[3] Fleet Battle Experiment Echo, Asymmetric Urban Threat, `http://www.nwdc.navy.mil/navigation/mbc.htm`, last modified 8/31/00, last accessed 9/17/00.

[4] In this paper, we use K and M to mean powers of two (10 and 20, respectively) and k and m to mean powers of 10 (3 and 6, respectively). Thus kbps means 10^3 bits per second.

dynamic properties of this system, such as the buffer requirements (queue lengths).

3.1. The parameters

Below we define all of the parameters used in our model, for easy reference:

- d = input data streams' speed (bits/s);
- t = time interval (s);
- $D = td$, the size of a data chunk arriving during time period t (bits);
- B = wireless channel's total physical bandwidth (bits/s);
- β_b = communication overhead factor for broadcast ($\beta_b < 1$);
- $B_b = B\beta_b$, the effective bandwidth available for broadcast (bits/s);
- β_a = communication overhead factor for agents ($\beta_a < 1$);
- $B_a = B\beta_a$, the effective bandwidth available for agent messages (bits/s);
- B_I = the bandwidth available in the server's wired Internet connection, for receiving data streams (bits/s); presumably $B_I \gg B$;
- n = number of client machines;
- i = index of a client machine ($1 \leqslant i \leqslant n$);
- m_i = number of tasks on each client machine i, $1 \leqslant i \leqslant n$;
- j = index of a task ($1 \leqslant j \leqslant m_i$);
- $m = \sum m_i$, total number of tasks;
- r = arrival rate of new agents uploaded from the clients to the server (per second);
- K = average agent size (bits);
- F'_{ij} = the fraction of the total data D that task j on client i chooses to process (by choosing to process only certain data streams);
- F_{ij} = the fraction of the data processed by task j on client i, produced as output;
- $c_{ij}(D, F'_{ij}, F_{ij})$ = computational complexity of task j on client i (operations);[5]
- μ = the average computational complexity, for a given D ($\mu = (1/m)\sum_{ij} c_{ij}(D, F'_{ij}, F_{ij})$); it is a convenient shorthand;
- C_{init} = average number of operations needed for a new agent to start and to exit;
- S_i^c = performance of client machine i (operations/s);
- α_i^c = performance efficiency of the software platform on the client machine i ($\alpha_i^c < 1$);
- S^s = performance of the server machine (operations/s);[6]
- α^s = performance efficiency of the software platform on the server ($\alpha^s < 1$).

[5] We expect that $c(\,)$ will have little dependence on D, directly, but more on DF'_{ij}.

[6] We assume that all agents get equal-priority access to server cycles.

Notes. B is the raw bandwidth of the wireless channel, but that bandwidth is never fully available to application communication. We assume that a broadcast protocol would actually achieve bandwidth B_b and a mobile-agent messaging protocol would achieve bandwidth B_a after including the overhead of protocol and retransmissions. In section 4 we discuss our measurements of B_a and B_b.

When comparing a mobile-agent approach to a more traditional approach, it is most fair to expect that a traditional system would use compiled code on the client (such as compiled C code), whereas a mobile-agent system would use interpreted code on the server (because most mobile-agent systems only support interpreted languages like Java or Tcl). The client and server will likely be different hardware and have different speeds, S^c and S^s, respectively. Because the language, compiler, and run-time system impose overhead, the client runs at a fraction α^c of the full speed S^c, and the server runs at a fraction α^s of the full speed S^s. Of course, $\alpha < 1$, and we expect $\alpha^s < \alpha^c$, since filtering agents on the server will be interpreted, whereas filtering code on the clients will be compiled. On the other hand, we expect $S^s \gg S^c$.

Computed values. As hinted in the figures above, the following values are computed as a result of the other parameters:

- T_I: the time for transmission across the Internet to the server;
- T_S: the time for processing on the server;
- T_W: the time for transmission across the wireless network;
- T_C: the time for processing on the client.

Most of these have two variants, i.e., T_{SA}, T_{WA} and T_{CA} for the agent case, and T_{SB}, T_{WB} and T_{CB} for the broadcast case.

3.2. Computing the constraints

As mentioned above, each stage of the pipeline must complete in less than time t, that is, $T_I \leqslant t$, $T_S \leqslant t$, $T_W \leqslant t$, and $T_C \leqslant t$.

Internet, T_I. Since we are concerned with alternatives for the portion of the system spanning the wireless network, we do not specifically model the Internet portion. We assume that the Internet is not the bottleneck, that is, it is sufficiently fast to deliver all data streams on schedule:

$$T_I = \frac{D}{B_I} \leqslant t, \tag{1}$$

$$d \leqslant B_I, \tag{2}$$

of course.

Server, T_S. In the broadcast case, the server simply merges the data streams arriving from the Internet. This step is trivial, and in any case $T_{SB} < t$ almost certainly.

In the agent case, data filtering happens on the server. The server's time is a combination of the filtering costs plus the time spent initializing newly arrived agents:

$$T_{SA} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \frac{c_{ij}(D, F'_{ij}, F_{ij})}{\alpha^s S^s} + \frac{rtC_{init}}{\alpha^s S^s}. \tag{3}$$

If we know that the expected value of the computing complexity c_{ij} is μ, then we can simplify and obtain a bound on the number of client tasks (agents), m. That is, we assume that

$$\frac{\sum_{i,j} c_{ij}(D, F'_{ij}, F_{ij})}{\alpha^s S^s} \approx \frac{m\mu}{\alpha^s S^s}. \tag{4}$$

Now $T_{SA} \leqslant t$,

$$\frac{m\mu + rtC_{init}}{\alpha^s S^s} \leqslant t, \tag{5}$$

$$m \leqslant \left(\alpha^s S^s - rC_{init}\right) \frac{t}{\mu}. \tag{6}$$

Wireless network, T_W. The broadcast case is relatively simple, since all of the chunk data D is sent over the channel:

$$T_{WB} = \frac{D}{B_b} \leqslant t, \tag{7}$$

$$d \leqslant B_b. \tag{8}$$

Recall that $B_b = B\beta_b$, and that $D = td$.

In the agent case, agents filter out most of the data and send a subset of the data items across the wireless network, as messages back to their task on the client. Agent_{ij} sends, on average, $DF'_{ij}F_{ij}$ bits from a chunk. The total time to transfer all agents' messages is, thus,

$$T_{WA} = \frac{\sum_{i,j} DF'_{ij}F_{ij}}{B_a} \leqslant t. \tag{9}$$

If we consider the average agent and define

$$F'F \equiv \frac{1}{m} \sum_{i,j} F'_{ij}F_{ij}, \tag{10}$$

then, since there are m agents,

$$\frac{mDF'F}{B_a} \leqslant t. \tag{11}$$

It is not quite that simple, however.

The wireless channel also carries agents from the clients to the server, so we must adjust for the bandwidth occupied by traffic in the reverse direction.[7] Recall that new agents of size K jump to the server at a rate r per second. This activity adds rK bits per second (rtK bits per chunk) to the total traffic. So, updating equation (11) we have

$$\frac{mDF'F + rtK}{B_a} \leqslant t, \tag{12}$$

which leads to a bound on the number of agents (tasks):

$$m \leqslant \frac{B_a - rK}{dF'F}. \tag{13}$$

When does the mobile-agent approach require less wireless bandwidth? We can compute the bandwidth needed from the amount of data transmitted for one chunk, expanded by $1/\beta$ to account for the protocol overhead, then divide by the time t for one chunk:

$$\frac{1}{t}\left(\frac{1}{\beta_a}\left(mDF'F + rtK\right)\right) < \frac{1}{t}\left(\frac{1}{\beta_b} D\right), \tag{14}$$

$$mdF'F + rK < \frac{\beta_a}{\beta_b} d, \tag{15}$$

$$m < \frac{1}{F'F}\left(\frac{B_a}{B_b} - \frac{rK}{d}\right). \tag{16}$$

Note that inequality (16) is nearly the same as inequality (13). If broadcast is possible ($d \leqslant B_b$), then we should use broadcast iff m exceeds the limit provided in inequality (16). If broadcast is impossible ($d > B_b$), then of course the mobile-agent approach is the only choice, but the number of agents must be kept within the limit specified in (13).

Note that in the broadcast case the wireless bandwidth must scale with the input stream rate, while in the agent case the wireless bandwidth must scale with the number of agents and the relevance of the data. Since we expect that most of the data will be filtered out by agents (i.e., $F'F < 0.01$), the agent approach should scale well to systems with large dataflow rates and moderate client populations.

Client, T_C. We consider only the processing needed to filter the data stream, and assume that the clients have additional power and time needed for an application-specific consumption of the data. Also, we assume the client has sufficient processing power to launch agents at rate r/n.

In the broadcast case, the data filtering happens on the clients. We must design for the slowest client, i.e.,

$$T_{CB} = \max_i \sum_{j=1}^{m_i} \frac{c_{ij}(D, F'_{ij}, F_{ij})}{\alpha_i^c S_i^c}. \tag{17}$$

If all n client hosts were the same, we could write simply

$$T_{CB} = \frac{m}{n} \frac{\mu}{\alpha^c S^c}, \tag{18}$$

and since $T_{CB} \leqslant t$ is required,

$$m \leqslant n\alpha^c S^c \frac{t}{\mu}. \tag{19}$$

In the agent case there is no data filtering on the clients, so $T_{CA} = 0$.

[7] Unless the channel is full duplex, in which case there is no impact on the downlink bandwidth. Here we assume a half-duplex channel.

Table 1
Summary of the constraints derived earlier, along with simplified constraints that assume $r = 0$. T_I and T_C are not affected by r. At the bottom, we show the comparison where agents require less wireless bandwidth than the broadcast approach.

Stage	Limits		Simplified limits	
	Broadcast	Agent	Broadcast	Agent
Internet, T_I	$d \leqslant B_I$	$d \leqslant B_I$	$d \leqslant B_I$	$d \leqslant B_I$
Server, T_S	negligible	$m \leqslant \left(\alpha^s S^s - rC_{init}\right)\frac{t}{\mu}$	negligible	$m \leqslant \left(\alpha^s S^s\right)\frac{t}{\mu}$
Wireless, T_W	$d \leqslant B_b$	$m \leqslant \frac{B_a - rK}{dF'F}$	$d \leqslant B_b$	$m \leqslant \frac{B_a}{dF'F}$
Client, T_C	$m \leqslant n\left(\alpha^c S^c\right)\frac{t}{\mu}$	negligible	$m \leqslant n\left(\alpha^c S^c\right)\frac{t}{\mu}$	negligible
	Comparison		Simplified comparison	
Wireless, T_W	$m < \frac{1}{F'F}\left(\frac{B_a}{B_b} - \frac{rK}{d}\right)$		$m < \frac{1}{F'F}\frac{B_a}{B_b}$	

3.3. Commentary

The results are summarized in table 1.

We can see that the agent approach fits within the constraints of the wireless network if the number (m and r) and size (K) of agents is small, or the filtering ratios ($F'F$) are low.

We believe that, in many realistic applications, most agents will remain on the server for a long time, and new agents will be installed rarely. Thus, r is small. Most of the time, $r = 0$. This assumption simplifies some of the equations into a more readable form, as shown in the right side of the table.

Notice that the broadcast case scales infinitely with the number of clients, but to add tasks to a client or to add data to the input stream requires the client processor to be faster. On every client i

$$\sum_{j=1}^{m_i} \frac{c_{ij}(D, F'_{ij}, F_{ij})}{\alpha_i^c S_i^c} \leqslant t, \tag{20}$$

$$S_i^c \geqslant \sum_{j=1}^{m_i} \frac{c_{ij}(D, F'_{ij}, F_{ij})}{\alpha_i^c t}, \tag{21}$$

so, as d or t increases or as m_i (the range of j) increases, S_i^c must increase.

The mobile-agent case, on the other hand, requires little from the client processor (for filtering), but requires a lot more from the server processor. That processor must scale with the input data rate, the number of clients, and the number of tasks per client:

$$S^s \geqslant \frac{m\mu + rtC_{init}}{t\alpha^s}. \tag{22}$$

On the other hand, it may be easier to scale a server in a fixed facility than to increase the speed of individual client machines, especially if the server lives in a comfortable machine room while the clients are mobile, battery-operated field machines.

Buffers in the pipeline. Since we model our application as a pipeline, we are primarily concerned with throughput and bandwidth, rather than response time and latency. As long as the pipeline is stable in the steady state, i.e., no component's capacity is exceeded, the system works. All of our above calculations are based on that approach.

In a real system, of course, the data flow fluctuates over time. Buffers between each stage of the pipeline hold data when one stage produces data faster than the next stage can process it. In a more complete analysis we would use a full queuing model to analyze the distribution of buffer sizes at each stage of the pipeline, given distributions for parameters like d, r, and $c()$. We leave this analysis for future work.

Latency. Although we are most concerned with throughput, in our application some clients may also be concerned about latency. In other words, it would be a shame if time-critical data were delayed from reaching the client. Which approach leads to less latency, say, from the time it reaches the server until the time it reaches the client application? Consider the flow of a specific data item through the pipeline: it is processed on the server, transmitted on the wireless network, and processed on the client. It must share each of these resources with other data items in its chunk, and it must share the server and wireless network with other clients. On average, each of m agents may require only T_{SA}/m CPU time on the shared server. If the server divides its time finely and evenly, all tasks will complete their computation at time T_{SA}. If the server divides its time coarsely, the average task completes in half that time, at time $T_{SA}/2$. A similar analysis can be made for the wireless network.

Assuming fine-grain sharing of the server and network, the latencies are

$$L_A = T_{SA} + T_{WA} + T_{CA}, \tag{23}$$

$$L_B = T_{SB} + T_{WB} + T_{CB}. \tag{24}$$

If we ignore the arrival of new agents (i.e., $r = 0$), and assume that all clients are identical, we have

$$L_A = \frac{m\mu}{\alpha^s S^s} + \frac{mDF'F}{B_a} + 0, \tag{25}$$

$$L_B = 0 + \frac{D}{B_b} + \frac{m\mu}{n\alpha^c S^c}. \tag{26}$$

Unfortunately, it is difficult to compare these two without specific parameter values.

We wonder, however, about the value of such a latency analysis. Given a specific data rate d, one must choose a server speed, wireless network bandwidth, and client speed, that can just keep up with the data flow. That is, in time interval t two of those three components must each be able to process D data. Their combined latency is $2t$. With sufficiently small t, say, 1–10 s, it seems likely this latency would suffice for most applications. Although one approach may have a little less latency than the other, the data flow rate remains the same. One could reduce latency by making balanced improvements to the two components with non-zero latency; this improvement may be easier in the agent approach, because it may be easier to upgrade the server than thousands of clients.

4. Model parameters

To explore some of the performance space represented by the model, we need reasonable values for key parameters. In the context of the Fleet Battle Experiment (section 2) we have $B = 768$ kbps and $K = 8192$ bits. Using the extrapolation to a realistic situation, FBE-Echo anticipates $F'F = 0.005$, $r = 10/\text{s}$, and $d = 600$ kbps (based on 5 sensors, each generating 100 reports per second, at 150 bytes per report). This combination of parameters is realistic and conservative, in that it allows the broadcast approach to succeed (because $d < B$).[8] Let us presume that an average agent monitors 2 of the 5 sensors, that is, $F' = 0.4$. Thus, if we accumulate reports for a $t = 10$ s interval, the total amount of data $D = 6$ mbits, and the agent must process $DF' = 2.4$ mbits.

Unfortunately, CAST was not instrumented to record any specific information about the computational cost or software overhead α of the CAST agents. Thus, to measure the value of the other model parameters, we constructed a small test environment consisting of two Linux laptops, a Linux workstation cluster, and a wireless network. One laptop served as the wireless client machine. The other laptop ran `routed` to serve as a gateway between the 2 Mbps wireless network and the 10 mbps wired network. Our server cluster contained 14 Linux workstations. We treated the 14 machines as a single logical server, because we needed that many to effectively measure β_{a}, as we describe below. The platform can be envisioned as shown in figure 4.[9]

[8] In practice, the SHF satellite network was congested due to other traffic; although we do not model this congestion, it would only lead to a stronger case for mobile agents since they require less bandwidth.

[9] Client: Gateway Solo 2300 laptop; Intel Pentium MMX 200 MHz, 48 MB RAM, running Linux 2.0.36. Gateway: Tecra 500CS laptop; Intel Pentium 120 MHz, 16 MB RAM, running Linux 2.2.6. Servers: VA Linux VarStation 28, Model 2871E; Pentium II at 450 MHz, 512K ECC L2 Cache, 256 MB RAM, running Linux 2.0.36. Wired network: the gateway was connected to a 10 mbps Ethernet, through a hub, a 10 mbps switch, and a 100 mbps switch, to the server cluster. Wireless network: 2 Mbps Lucent WaveLAN "Bronze Turbo" 802.11b PC cards configured at 2 Mbps.

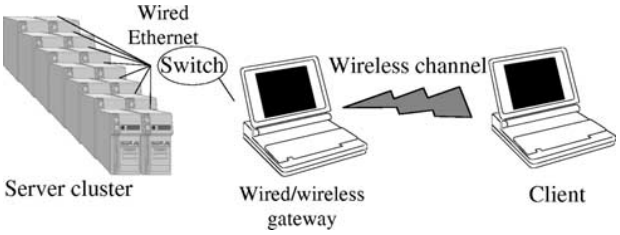

Figure 4. The experimental platform, in which the server is a cluster of workstations, sending its data through a wireless gateway machine to the wireless network.

When measuring parameters related to mobile agents, we used the Dartmouth mobile-agent system D'Agents [6,7] as an example of a canonical mobile-agent platform. Although the CAST application used a different mobile-agent platform, it was sufficiently similar to D'Agents for the purposes of the experiments here.

4.1. Measuring α

Because the language, compiler, and run-time system impose overhead, the client runs at a fraction α^{c} of the full speed S^{c}, and the server runs at a fraction α^{s} of the full speed S^{s}. Unfortunately, we do not know and cannot directly measure S.[10] On a single host of speed S, though, we can run a compiled C program and a comparable Java program, to obtain $\alpha^{\text{c}}S$ and $\alpha^{\text{s}}S$, and divide to obtain $\alpha^{\text{c}}/\alpha^{\text{s}}$.

We wrote a simple image-processing application (an edge detector) in C, and then ported it to Java. We ran them both on one of our servers, using a sample image;[11] averaged over 100 runs, the Java program took 111 ms and the C program took 83 ms. In this measurement, we include only the computational portion of the application, rather than the time to read and write the image files, since in our modeled application the data will be streaming through memory, rather than on disk. These numbers give $\alpha^{\text{s}}/\alpha^{\text{c}} = 0.75$, i.e., C was 25% faster than Java.

4.2. Measuring β

The raw bandwidth of our WaveLAN wireless network was 2 Mbps (that is, 2,097,152 bps). To obtain β values, we measured the transmission speed of sample applications transmitting data across that network, and divided by 2 Mbps.

To compute β_{b} for the broadcast case, we wrote a simple pair of programs; one broadcast 4999 data blocks of 50,000 bytes each across the wireless link, for the other to receive. The transmission completed in 1135 s, which implies that

$$B_{\text{b}} = \frac{4999 \times 50{,}000\ \text{Bytes} \times 8\ \text{bits/Bytes}}{1135\ \text{s}}, \tag{27}$$

$$\beta_{\text{b}} = \frac{B_{\text{b}}}{B} = \frac{1{,}761{,}762\ \text{bps}}{2{,}097{,}152\ \text{bps}} = 0.840. \tag{28}$$

[10] Recall the difficulty of measuring the "peak performance" of an architecture, and all the discussions about the value of MHz and MIPS as metrics of performance.

[11] The image size was 308,378 bytes, or 2,467,024 bits, approximately the value $DF' = 2.4$ mbits we mentioned above.

In other words, broadcast of these reasonably large chunks of data is 84% efficient.

To compute β_a for the agent case, we wrote a simple agent program that visits the server, and sends about 50 KB of documents every 3 s. The agent completes after sending 500 of these 50 KB messages. The effective bandwidth is computed as the total amount of data transmitted divided by the time required to transmit the data, including the time sleeping. To better reflect the modeled application, we actually sent out several agents to different hosts within our server cluster, and increased the number of agents and hosts until we reached the highest possible total bandwidth. We found that 14 agents, running on separate hosts within the server cluster, reached almost 1.5 mbps. Specifically,

$$\beta_a = \frac{B_a}{B} = \frac{1{,}484{,}144 \text{ bps}}{2{,}097{,}152 \text{ bps}} = 0.708. \tag{29}$$

4.3. Measuring C_{init}

When hosting agents, the server needs to support all of their computational needs. In addition to the processing time required to filter the data, new agents come and old agents exit. In our model, r agents come and go, per second, on average. We model the computational overhead of each agent's start and exit as C_{init}. We wrote a trivial agent and arranged for one of our server hosts to rapidly submit agents to another server host. After 5000 submit/exit pairs in 204 s, we conclude that the overhead C_{init} is about 40 ms (actually, it is the number of operations corresponding to 40 ms). It may be less, because our measurement was based on wall-clock time, not CPU time, and this experiment did not max out the CPU.

5. Results

We now use these parameters in our equations to get a sense of how they react under specific conditions.

Unfortunately, it is difficult to get actual μ, α, and S parameters, although we did measure some ratios above. If we assume, however, that our edge-detection algorithm is representative of one sort of filtering operation, we do know the time it took to execute that operation. On our client laptop we measured

$$\frac{\mu}{\alpha^c S^c} = 236 \text{ ms}. \tag{30}$$

That represents the time needed to process one 308 KByte (precisely, 2,467,024 bit) image; that is, approximately $DF' = 2.4$ mbits for $t = 10$ s. Equation (19) tells us that

$$\frac{m}{n} \leqslant \alpha^c S^c \frac{t}{\mu} = \frac{10}{0.236} = 42, \tag{31}$$

that is, about 42 tasks per client, for an arbitrary number of clients n. Seen another way, if the filter requires 2.36% (236 ms of the 10 s interval) of the client's CPU, the client could support 42 such filters. Of course, the client machine should reserve some power for consuming the data after filtering, so it should not run anywhere close to 42 filters.

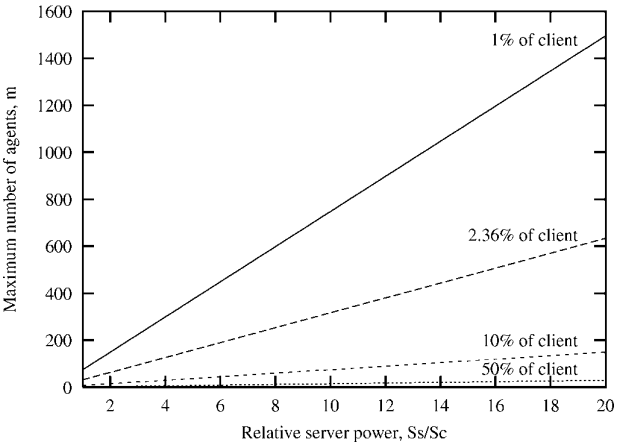

Figure 5. The number of agents that can effectively be supported, as the server power grows relative to the client's power. We show four curves, representing different possible computations; 2.36% represents our image-processing sample application. In the broadcast case, each client could support 100 tasks (1%), 42 tasks (2.36%), 10 tasks (10%), or 2 tasks (50%), for an arbitrary number of clients.

Similarly, on the server, if we ignore r, equation (6) tells us that

$$m \leqslant \left(\alpha^s S^s\right) \frac{t}{\mu}. \tag{32}$$

The machines we used as "servers" in our experiments were not particularly speedy. It is more interesting to derive an equation for m in terms of the relative power of the server and client, using quantities that we already have measured:

$$\begin{aligned} m \leqslant \frac{\alpha^s S^s}{\mu} t &= \frac{\alpha^c S^c}{\mu} \frac{\alpha^s}{\alpha^c} \frac{S^s}{S^c} t \\ &= \frac{1}{0.236 \text{ s}} (0.75) \frac{S^s}{S^c} (10 \text{ s}) = 31.7 \frac{S^s}{S^c}. \end{aligned} \tag{33}$$

Figure 5 shows the total number of agents (for all clients) that could be supported as the power of the server S^s grows relative to the power of the clients S^c, for our 236 ms sample task as well as three other possibilities. The plot shows ratios S^s/S^c reaching up to 20, which is easily obtainable when clients are portable computers.

In figure 6 we show the constraints on m, in the agent case. This graph plots the two constraints from table 1, as d varies. The actual constraint is the minimum of the two curves. For lower $F'F$, the server's computation is the tighter constraint; for higher $F'F$, the wireless network bandwidth limits us more. As a basis for drawing these curves, we reconsider the example inspired by FBE-Echo – that is, $d = 600$ kbps, $t = 10$ s, and $F' = 0.4$ – and measure the image-processing application running on the server ($\mu/\alpha^s S^s = 111$ ms, as described earlier). Of course, in nearly any application μ will vary with D (and thus, with d and t); for the purposes of this illustrative graph we assume the computation is linear. In other words, we imagine that μ may behave as follows:

$$\frac{\mu}{\alpha^s S^s} = 111 \text{ ms} \times \frac{D}{10 \text{ s} \times 600 \text{ kbps}}. \tag{34}$$

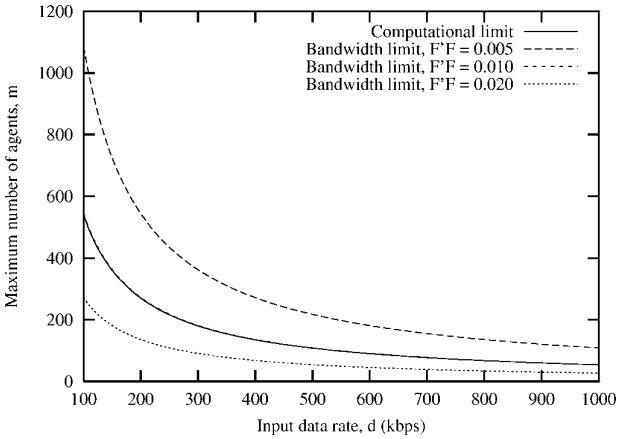

Figure 6. The maximum number of agents m we can support, given the constraints in table 1. Here $B = 768$ kbps, $r = 0$, $\beta_a = 0.708$, $t = 10$ s, and μ is proportional to D, as described in the text. The computational limit is coincidentally the same as the bandwidth limit for $F'F = 0.010$.

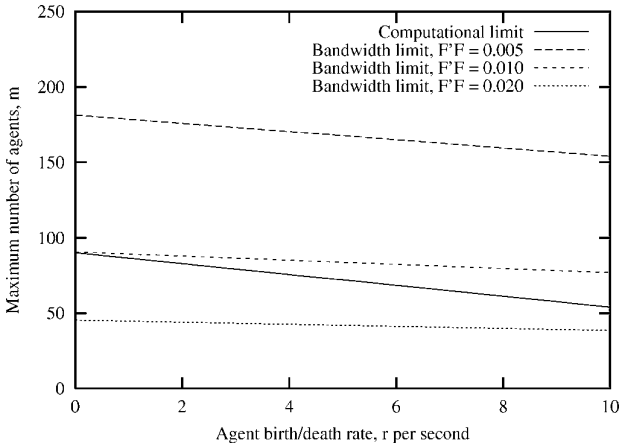

Figure 7. The maximum number of agents m we can support, given the constraints in table 1, as we vary r. Here we use parameters $K = 1$ KByte, $B = 768$ kbps, $d = 600$ kbps, $\beta_a = 0.708$, $t = 10$ s, $C_{init}/(\alpha^s S^s) = 40$ ms, and $\mu/(\alpha^s S^s) = 111$ ms.

In figure 7 we look at similar results when we vary r (the previous graph assumed $r = 0$). In section 4.3 we measured

$$\frac{C_{init}}{\alpha^s S^s} = 40 \text{ ms}, \tag{35}$$

and in section 4.1 we measured

$$\frac{\mu}{\alpha^s S^s} = 111 \text{ ms}, \tag{36}$$

and for a fixed $t = 10$ s, the computational constraint from equation (6) is

$$\begin{aligned} m &\leqslant \left(\frac{\alpha^s S^s}{\mu} - \frac{r C_{init}}{\mu}\right) t \\ &= \left(\frac{1}{111 \text{ ms}} - r\,\frac{40 \text{ ms}}{111 \text{ ms}}\right)(10 \text{ s}). \end{aligned} \tag{37}$$

Again, the actual constraint is the minimum of the two curves. For lower $F'F$, the server's computation is the tighter constraint; for higher $F'F$, the wireless network bandwidth limits us more.

In figure 7 the bandwidth-constraint lines are close to horizontal, since in FBE-Echo the 1 KByte agents are small enough that the transmission of the agents does not have a significant effect on the wireless network. As shown in figure 8, on the other hand, the behavior is dramatically different when the agents are larger. For large agents and high birth/death rates, the traffic induced by the jumping agents (rK) consumes the available bandwidth B_a, leaving nothing for agents to transmit their data. Clearly, such a system can support few agents when the agent size is large or when the birth/death rate is high. This result emphasizes the need to include some kind of code caching in any mobile-code system, so that the same agent code is not transmitted repeatedly across the wireless link.

Another useful way to look at the results is to graph the bandwidth required by either the agent approach or the broadcast approach, given certain parameters. In figure 9 we vary the filtering ratio, since it clearly has a large impact on the

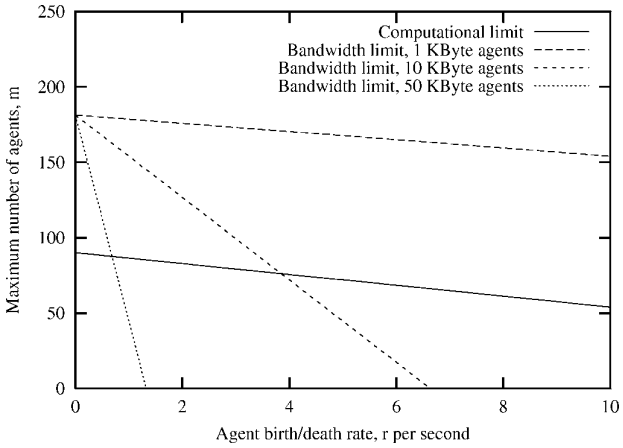

Figure 8. For comparison with figure 7, we fix $F'F = 0.005$ as in FBE-Echo, and instead display the bandwidth limit with $K = 1$, 10, or 50 KBytes. Other parameters are the same as in the preceding figure.

bandwidth required by the agent approach. For low filtering ratios, the agent approach needs less bandwidth than the broadcast approach. If $d > B$ (not shown), of course the broadcast approach cannot work at all, and the agent approach is the only solution.

In figure 10 we show the relationship between the number of agents and the necessary filtering ratio. Another view on the earlier charts, this clearly shows that, to support a large number of agents, those agents must be aggressively filtering the input stream.

In all, it is clear that there is a wide range of situations in which mobile agents are more efficient than the broadcast approach. Unless the number of clients is very large, the filtering ratio of each task is high, or the size or computational demands of each task is high, the mobile-agent approach has promise.

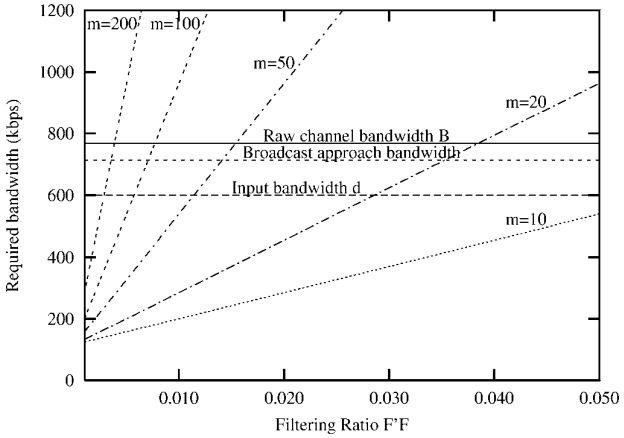

Figure 9. The bandwidth requirements for agent and broadcast approaches. Here $d = 600$ kbps, $B = 768$ kbps, $r = 10/\text{s}$, $K = 1$ KByte, $\beta_\text{a} = 0.708$, and $\beta_\text{b} = 0.840$. Note that the bandwidth required by the broadcast approach is d/β_b, and appears above d.

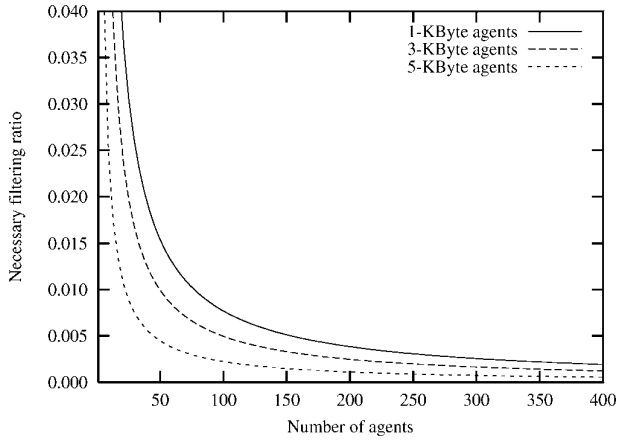

Figure 10. The relationship between the filtering ratio and the number of agents in the server. Other parameters are as before: $B = 768$ kbps, $d = 600$ kbps, $K = 1$ KByte, $r = 10/\text{s}$.

6. Related work

Performance modeling of computer networks and distributed applications is an old field, and our approach and resulting equations are similar to many previous analyses of distributed systems [10]. In addition, there has been some similar modeling work specifically for *mobile-agent* systems.

Strasser and Schwehm [16] develop a general model for comparing the performance of Remote Procedure Calls (RPC) with the performance of migrating agents. Using their model, which is best-suited for information-retrieval applications, they derive equations for the total number of bytes transferred across the network, as well as the total completion time of the task. The equations include such parameters as the expected result size and the "selectivity" of the agent (i.e., how much irrelevant information the agent filters out at the data site, rather than carrying with it for future examination). Their byte equations are similar to our bandwidth equations, although their time equations are not directly applicable to our scenario, since we are interested only in whether the server can keep up with the incoming data streams, not with the total completion time.

Küpper and Park [11] examine a signaling application inside a telecommunications network, and compare a mobile-agent approach with a stationary-agent (or client–server) approach. Starting with a queuing model of a hierarchical signaling network, they produce equations that specify the expected load on each network node in both the mobile and stationary cases. These equations are similar to our server-load equations (from which we derive the constraint on how many agents the server machine can handle simultaneously).

Picco, Fuggetta and Vigna [4,12] identify three main design paradigms that exploit code mobility: remote evaluation, code on demand, and mobile agents. Within the context of a network-management application, i.e., the polling of management information from a pool of network devices, they analyze these three paradigms and the traditional client–server paradigm. They develop analytical models to compare the amount of traffic around the network-management server, as well as the total traffic on the managed network. These models are similar to our bandwidth models.

More recently, Puliafito et al. [13] used Petri nets to compare the mobile-agent, remote-evaluation and client–server paradigms. The key parameters to the models are transition probabilities that specify (1) whether a traditional client or agent will need to redo an operation, and (2) whether a client or agent will need to perform another operation to continue with the overall task. Using the models, they compare the mean time to task completion for the three paradigms. Like the the work of Strasser and Schwehm [16], these Petri-net models are well suited for information-retrieval applications, are more general than the models in the other papers, and are not directly applicable to our scenario, which involves continuous filtering of an incoming data stream, rather than a multi-step retrieval task. Petri nets, however, could be a useful analysis technique for our scenario.

In addition to the mathematical analyses above, there has been a range of simulation and experimental work for mobile-agent systems. Recent simulation work includes [15], which considers the use of mobile agents for search operations on remote file systems (such as the standard substring search of the Unix *grep* command), and [1], which examines the use of mobile agents for message delivery in ad hoc wireless networks. Recent experimental work includes [14], which compares different strategies for accessing a Web database, and [5], which compares RPC and mobile-agent approaches for accessing a document database. Although we have not done simulation or experimental validation of our model yet, such validation is an essential part of future work.

In our broadcast scenario all of the data are broadcast. In our agent scenario each agent sends its own copy of the filtered data to its client, regardless of whether other clients may also want the data. We may be able to use techniques from the domain of "broadcast publishing" to obtain a more efficient compromise approach [9].

7. Summary and future work

The FBE-Echo experiment is a good example of an application in which only a small portion of the available information is relevant to the task at hand. Although the FBE-Echo experimental results suggested that mobile agents were achieving significant bandwidth savings, the aggressive FBE testing schedule did not allow a *direct* comparison with a traditional client/server implementation. For this reason, we developed the model described in this paper, which confirms that mobile agents will often have a significant performance benefit in filtering applications. With small filtering ratios ($F'F$) or a small numbers of agents, a mobile-agent approach can get by with less bandwidth or slower (i.e., cheaper or lighter) clients. At the same time, our analysis reinforces the importance of the engineering challenge of keeping α^{s} and β_{a} large, that is, reducing the overhead of mobile-agent computation and communication.

To further develop this performance analysis and to use it in a wider range of applications, we need to better understand several issues: How variable is the input data stream in terms of its flow rate? In other words, how much buffering would be necessary in the server and the clients? How many different agent/task types are there in typical applications, and how widely do these types vary? How much CPU time is needed to support the network protocols? Are average or expected numbers acceptable, or do we need worst-case analysis?

Furthermore, we need to address a few limitations: (1) the broadcast case assumes that nobody misses any transmissions, or that they do not care if they miss it, so there are no retransmissions; (2) both cases ignore the client processing consumed by the end application; and (3) we consider only one application scenario. While the application scenario is widely representative, there are certainly other application types worth analyzing. In particular, we would like to consider scenarios in which the mobile agents move up and down a hierarchy of gateway machines. We are also interested in the use of mobile agents as a dynamically distributed, and redistributed, cooperative cache to support mobile computers in a wireless network.

Acknowledgements

Many thanks for the helpful input from Daniela Rus, Jeff Bradshaw, Niranjan Suri, and the anonymous reviewers for MONET and for the Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000).

References

[1] S. Bandyopadhyay and K. Paul, Evaluating the performance of mobile agent-based message communication among mobile hosts in large ad hoc wireless network, in: *Proceedings of the Second ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (ACM Press, August 1999) pp. 69–73.

[2] D.A. Chacón, Small agents solving large problems, Lockheed Martin ATL Internal Report (1999) `http://www.atl.external.lmco.com/overview/library.html`

[3] D. Chess, C. Harrison and A. Kershenbaum, Mobile agents: Are they a good idea?, in: *Mobile Object Systems: Towards the Programmable Internet*, Lecture Notes in Computer Science, Vol. 1222 (Springer-Verlag, 1997) pp. 25–47.

[4] A. Fuggetta, G.P. Picco and G. Vigna, Understanding code mobility, IEEE Transactions on Software Engineering 24(5) (1998) 342–361.

[5] R.S. Gray, G. Cybenko, D. Kotz and D. Rus, Mobile agents: Motivations and state of the art, in: *Handbook of Agent Technology* (AAAI/MIT Press, 2002).

[6] R. Gray, Agent Tcl: A flexible and secure mobile-agent system, PhD thesis, Technical report TR98-327, Department of Computer Science, Dartmouth College (1997).

[7] R.S. Gray, Agent Tcl: A flexible and secure mobile-agent system, in: *Proceedings of the 1996 Tcl/Tk Workshop* (July 1996) pp. 9–23.

[8] M.O. Hofmann, A. McGovern and K.R. Whitebread, Mobile agents on the digital battlefield, in: *Proceedings of the Second Internal Conference on Autonomous Agents (AA'98)* (ACM Press, May 1998) pp. 219–225.

[9] T. Imielinski and S. Viswanathan, Wireless publishing: Issues and solutions, in: *Mobile Computing* (Kluwer Academic Publishers, 1996) chapter 11, pp. 299–329.

[10] P.J.B. King, *Computer and Communication System Performance Modeling* (Prentice Hall, 1990).

[11] A. Küpper and A.S. Park, Stationary vs. mobile user agents in future mobile telecommunications networks, in: *Proceedings of the Second International Workshop on Mobile Agents (MA'98)*, Lecture Notes in Computer Science, Vol. 1477 (Springer-Verlag, 1998).

[12] G.P. Picco, Understanding, evaluating, formalizing and exploiting code mobility, PhD thesis, Dipartimento di Automatica e Informatica, Politecnico di Torino (1998).

[13] A. Puliafito, S. Riccobene and M. Scarpa, An analytical comparison of the client–server, remote evaluation and mobile agents paradigms, in: *Proceedings of the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA99)* (IEEE Computer Society Press, October 1999).

[14] G. Samaras, M. Dikaiakos, C. Spyrou and A. Liverdos, Mobile agent platforms for web-databases: A qualitative and quantitative assessment, in: *Proceedings of the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA'99)* (IEEE Computer Society Press, October 1999) pp. 50–64.

[15] T. Spalink, J.H. Hartman and G. Gibson, The effects of a mobile agent on file service, in: *Proceedings of the First International Symposium on Agent Systems and Applications and Third International Symposium on Mobile Agents (ASA/MA'99)* (IEEE Computer Society Press, October 1999) pp. 42–49.

[16] M. Strasser and M. Schwehm, A performance model for mobile agent systems, in: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, Vol. 2 (June 1997) pp. 1132–1140.

David Kotz is an Associate Professor of Computer Science at Dartmouth College, Hanover, NH, where he has been since 1991. His research interests include mobile computing and wireless networks, mobile agents, parallel and distributed operating systems, multiprocessor file systems, and computer ethics. He received the M.S. and Ph.D. degrees in computer science from Duke University in 1989 and 1991, respectively. He received the A.B. degree in computer science and physics from Dartmouth College, Hanover, NH, in 1986. He is a member of the ACM, IEEE Computer Society, and USENIX associations, and of Computer Professionals for Social Responsibility.

E-mail: dfk@cs.dartmouth.edu

George Cybenko is the Dorothy and Walter Gramm Professor of Engineering at Dartmouth College. He joined Dartmouth's Thayer School of Engineering in the fall of 1992. Prior to joining Dartmouth, he held positions at Tufts University and the University of Illinois at Urbana-Champaign. Dr. Cybenko received the B.Sc. degree in mathematics from the University of Toronto and the M.Sc. and Ph.D. degrees in applied mathematics of electrical and computer engineering from Princeton University in 1978. Dr. Cybenko was the Kloosterman Distinguished Visiting Professor at Leiden University, the Netherlands, in 1996. He is the founding Editor-in-Chief of IEEE/AIP Computing in Science and Engineering. Dr. Cybenko has pioneered research in several areas: signal processing, parallel computing, neurocomputing and mobile agent systems. His current research interest is distributed information systems.
E-mail: George.Cybenko@dartmouth.edu

Robert S. Gray is a Research Engineer at the Institute for Security Technology Studies (ISTS) and at the Thayer School of Engineering, both at Dartmouth College. His Thayer School work focuses on the performance, scalability and analysis of mobile-code systems, as well as the use of mobile-code technology in military command-and-control applications. He is the lead programmer on D'Agents, one of the first mobile-agent systems, and is one of the lead software developers on two large mobile-code projects funded by DoD and by DARPA. In his other life at ISTS, Dr. Gray focuses on information retrieval tools for law-enforcement personnel. He holds a B.S. in computer science from the University of Vermont and a Ph.D. in computer science from Dartmouth College. He is a member of IEEE, ACM and USENIX.
E-mail: Robert.S.Gray@dartmouth.edu

Guofei Jiang is a Senior Research Engineer in the Institute for Security Technology Studies at Dartmouth College. He received his B.Sc. and Ph.D. degrees in Electrical and Computer Engineering from Beijing Institute of Technology in 1993 and 1998, respectively. From 1998 to 2000, he was a postdoctoral researcher in the Thayer School of Engineering at Dartmouth College. He currently focuses his research on network security, distributed computing, mobile agents and machine learning. He was certified by SANS GCIH network security certification and is a member of SANS GIAC advisory board.
E-mail: Guofei.Jiang@dartmouth.edu

Ronald A. Peterson is a Senior Programmer for the D'Agents Mobile Agent Project in the Computer Science Department of Dartmouth College since 1998. He is also the owner of Peterson Enterprises, which develops PC-based MIDI musical instruments and graphics software. During the sixteen years prior to 1998 he held several software engineering positions in industry including software quality-assurance consultant, lead QA Engineer on a widely used industrial graphic interface builder at Dataviews Corporation, and developer of electronic-countermeasures systems at Sanders Associates/Lockheed-Martin. His research interests include mobile agents and machine-vision interfaces for novel musical instruments. He received a B.A. degree in physics from Lawrence University in 1979. He is a member of the IEEE Computer Society.
E-mail: rapjr@cs.dartmouth.edu

Martin O. Hofmann is currently a Principal Member of the Engineering Staff in the Artificial Intelligence Lab at Lockheed Martin Advanced Technology Laboratories (ATL). He holds a Ph.D. in electrical engineering and AI from Vanderbilt University and a Dipl.-Ing. degree from the Technical University Vienna, Austria. At ATL he leads applied research and development on intelligent mobile agents and information fusion. He is a member of the ACM, IEEE, and AAAI.
E-mail: mhofmann@atl.lmco.com

Daria A. Chacón was an engineer with Lockheed Martin's Advanced Technology Laboratories. She served as a software and technical lead on several programs that deploy mobile agent applications to the military, and she was a chief contributor to the Extendable Mobile Agent Architecture (EMAA) agent system. She received Bachelor's degrees in computer science and psychology from Taylor University in 1996 and 1997.
E-mail: dachacon@bigfoot.com

Kenneth R. Whitebread is a Manager of AI Programs at Lockheed Martin Advanced Technology Laboratories. He received his Ph.D. in computer science from the University of Wisconsin in 1982. He currently oversees several research and development programs applying intelligent agent technology and related AI techniques to information processing and retrieval problems in military command and control as well as commercial environments.
E-mail: kwhitebread@atl.lmco.com

James Hendler is a Professor at the University of Maryland and the Director of Semantic Web and Agent Technologies at the Maryland Information and Networks Dynamics Lab. He has joint appointments in the Department of Computer Science, the Institute for Advanced Computer Studies and the Institute for Systems Research, and he is also an affiliate of the Electrical Engineering Department. Hendler was the recipient of a 1995 Fulbright Foundation Fellowship, is a member of the US Air Force Science Advisory Board, and is a Fellow of the American Association for Artificial Intelligence. He was recently a Chief Scientist of the Information Systems Office at the Defense Advanced Research Projects Agency (DARPA).
E-mail: hendler@cs.umd.edu