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

David Kotz
Guofei Jiang
Robert Gray
George Cybenko
Ronald A. Peterson

Dartmouth College

Computer Science Technical Report TR2000-366
May 2, 2000

**Abstract**

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

## 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, because 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 [CHK97, GCKR00].

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

Authors' addresses: Dartmouth College, Hanover NH 03755. Email: `First.Last@Dartmouth.edu`. The research was supported by the DARPA CoABS Program, contract F30602-98-2-0107, and by the DoD MURI program (AFoSR contract F49620-97-1-03821).

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 consumption from 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 pre-installed 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 network, 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 a typical application scenario. The scenario is sufficiently general to reflect many applications, from a military application in which field units are monitoring information sources as diverse as weather data and intelligence reports, to commercial applications in which consumers are monitoring 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, in a large system we expect that the server would be a large 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 regions or in overlapping regions. The overall picture is shown in Figure 1.

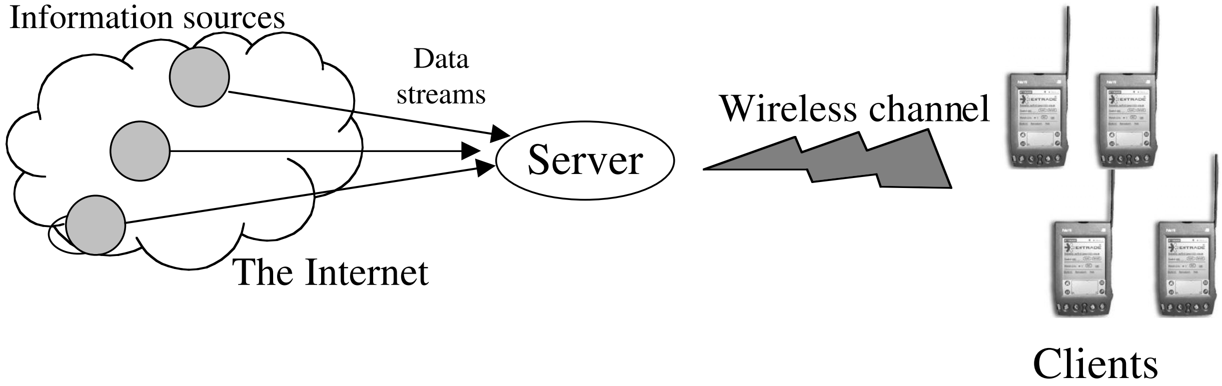


Figure 1: The scenario we analyze.

Each consumer is interested in a different (but not necessarily disjoint) subset of the data. The consumer is interested in only a few of the information streams, and then only in some filtered set of items in those streams. For example, a traveler may monitor the weather stream, but not the stock stream; of the weather stream, they may care only about the locations affecting their travels 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]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.

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.

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 it is important to keep client weight and power requirements (and cost) low.

In the next section, we list and define the parameters that arise in the analysis. After that, we derive the basic equations, and interpret their significance. In Section 3, we describe our experiments used to obtain the values of key parameters. In Section 4, we use the results of those experiments to explore the performance space given by our model. We describe some related work in Section 5, and summarize in Section 6.

## 2 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.

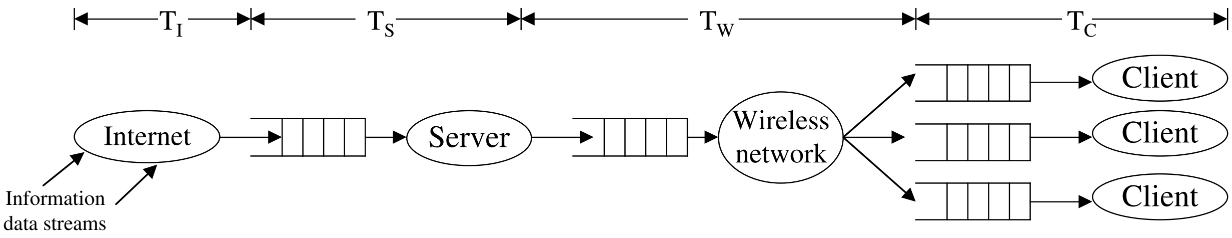


Figure 2: The scenario viewed as a pipeline.

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, $T_I \leq t$, $T_S \leq t$, $T_W \leq t$, and $T_C \leq t$. 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 dynamic properties of this system, such as the buffer requirements (queue lengths).

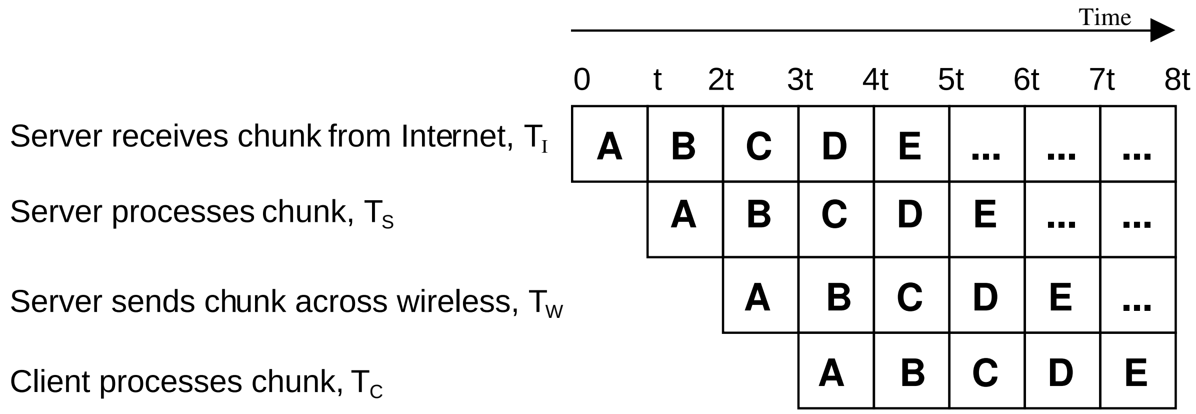


Figure 3: The pipeline timing diagram. The letters represent data chunks. For example, between time $3t$ and $4t$ chunk A is being processed by the clients, chunk B is being transmitted from the server to the clients, chunk C is being processed by the server, and chunk D is being received by the server.

## 2.1 The parameters

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

$d$ = input data streams' speed (bits/sec);
$t$ = time interval (seconds);
$D = td$, the size of a data chunk arriving during time period $t$ (bits);
$B$ = wireless channel's total physical bandwidth (bits/sec);
$\beta_b$ = communication overhead factor for broadcast ($\beta_b < 1$);
$B_b = B\beta_b$, the effective bandwidth available for broadcast (bits/sec);
$\beta_a$ = communication overhead factor for agents ($\beta_a < 1$);
$B_a = B\beta_a$, the effective bandwidth available for agent messages (bits/sec);
$B_I$ = the bandwidth available in the server's wired Internet connection, for receiving data streams (bits/sec); presumably $B_I >> B$;
$n$ = number of client machines;
$i$ = index of a client machine ($1 \le i \le n$);
$m_i$ = number of tasks on each client machine $i$, $1 \le i \le n$;
$j$ = index of a task ($1 \le j \le m_i$);
$m = \sum m_i$, 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);[2]
$\mu$ = the average computational complexity, for a given $D$ ($\mu = \sum_{ij} c_{ij}(D, F'_{ij}, F_{ij})$). 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/sec);
$\alpha_i^c$ = performance efficiency of the software platform on the client machine $i$ ($\alpha_i^c < 1$);
$S^s$ = performance of the server machine (operations/sec);[3]
$\alpha^s$ = performance efficiency of the software platform on the server ($\alpha^s < 1$);

**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 $B_a$. In Section 3 we discuss our measurements of $B_a$ and $B_b$.

When comparing a mobile-agent approach to a more traditional approach, we think 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 $\alpha^c$ of the full speed $S^c$, and the server runs at a fraction $\alpha^s$ of the full speed $S^s$. Of course $\alpha < 1$, and we expect $\alpha^s < \alpha^c$. On the other hand, we would expect $S^s >> S^c$.

**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., $T_{SA}$ for the agent case and $T_{SB}$ for the broadcast case, $T_{WA}$ for the agent case and $T_{WB}$ for the broadcast case, and $T_{CA}$ for the agent case and $T_{CB}$ for the broadcast case.

[2] We expect that $c()$ will have little dependence on $D$, directly, but more on $DF'_{ij}$.
[3] We assume that all agents get equal-priority access to server cycles.

## 2.2 Computing the constraints

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

**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:

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

$$d \leq B_I \tag{2}$$

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 $\mu$, then we can simplify and obtain a bound on the number of client tasks (agents), $m$. That is, we assume that

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

Now $T_{SA} \leq t$,

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

$$m \leq (\alpha^s S^s - rC_{init}) \frac{t}{\mu} \tag{6}$$

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

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

$$d \leq B_b \tag{8}$$

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

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

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

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

But it is not quite that simple.

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.[4] 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

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

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

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

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:

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

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

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

Note that inequality (16) is nearly the same as inequality (13). If broadcast is possible ($d \leq B_b$), 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 data-flow 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 $T_{CB} \leq t$ is required,

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

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

Table 1: Summary of the constraints derived earlier. There is only a meaningful comparison for the wireless channel.

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

## 2.3 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, Table 2.

Table 2: Simplified constraints, assuming $r = 0$. $T_I$ and $T_C$ do not change.

| | Limits | | Comparison |
|---|---|---|---|
| Stage | Broadcast | Agent | |
| Server, $T_S$ | 0 | $m \le (\alpha^s S^s)\frac{t}{\mu}$ | |
| Wireless, $T_W$ | $d \le B_b$ | $m \le \frac{B_a}{dF'F}$ | $m < \frac{1}{F'F}\frac{B_a}{B_b}$ |

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 $i$

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

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

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.

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

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

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

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 $\frac{1}{m}T_{SA}$ 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 $\frac{1}{2}T_{SA}$. 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 $t$ those three components must each be able to process $D$ data. Their latency is $3t$. With sufficiently small $t$, say, 1–10 seconds, 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.

# 3 Experiments to obtain parameters

To measure the value of several 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 $\beta_a$, as we describe below. The platform can be envisioned as shown in Figure 4.

## 3.1 Measuring $\alpha$

Because the language, compiler, and run-time system impose overhead, the client runs at a fraction $\alpha^c$ of the full speed $S^c$, and the server runs at a fraction $\alpha^s$ of the full speed $S^s$. Unfortunately, we do not know and cannot directly measure $S$.[5] On a single host of speed $S$, though, we can run a compiled C program and a comparable Java program, to obtain $\alpha^c S$ and $\alpha^s S$, and divide to obtain $\alpha^c/\alpha^s$.

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; averaging over 100 runs, the Java program took

[5]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.

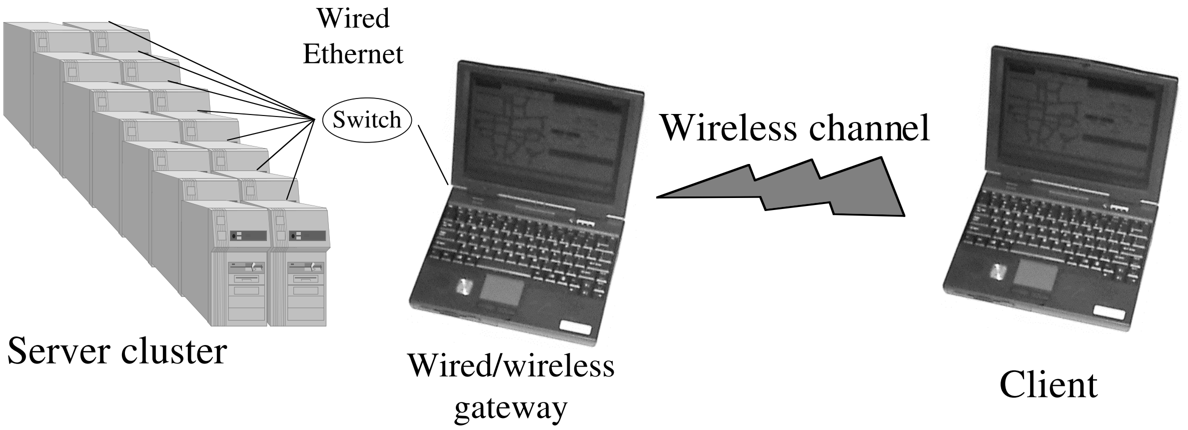


Figure 4: The experimental platform, in which the server is a cluster of workstations, sending its data through a wireless gateway machine to the wireless network. The details of the platform follow. Client: Gateway Solo 2300 laptop; Intel Pentium MMX 200 MHz, 48MB RAM, running Linux 2.0.36. Gateway: Tecra 500CS laptop; Intel Pentium 120 MHz, 16MB RAM, running Linux 2.2.6. Servers: VA Linux VarStation 28, Model 2871E; Pentium II at 450 MHz, 512K ECC L2 Cache, 256MB 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.

111 msec and the C program took 83 msec. In this measurement, we include only the computational portion of the application, rather than the time to read and write the image files, because in our modeled application the data will be streaming through memory, and not on disk. These numbers give $\alpha^s/\alpha^c = 0.75$, i.e., C was 25% faster than Java.

### 3.2 Measuring $\beta$

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

To compute $\beta_b$ 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 seconds, which implies that

$$B_b = \frac{4999 \times 50,000B \times 8b/B}{1135\,\mathrm{sec}} \tag{27}$$

$$\beta_b = \frac{B_b}{B} = \frac{1,761,762\,\mathrm{bps}}{2,097,152\,\mathrm{bps}} = 0.840 \tag{28}$$

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

To compute $\beta_a$ for the agent case, we wrote a simple agent program that visits the server, and sends about 50KB of documents every 3 seconds. The agent completes after sending 500 of these 50KB 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 about 1.4 Mbps. Specifically,

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

We use these constants in our equations to obtain the results below.

### 3.3 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 seconds, we conclude that the overhead $C_{init}$ is about 40 msec (actually, it is the number of operations corresponding to 40 msec). 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.

## 4 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 $\mu$, $\alpha$, 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

$$\frac{\mu}{\alpha^c S^c} = 236\,\text{msec} \tag{30}$$

If this computation represents the time needed for filtering data that arrived over $t = 10$ seconds, for example, Equation 19 tells us that

$$\begin{aligned} m/n &\leq \alpha^c S^c \frac{t}{\mu} && (31) \\ &= 10/0.236 && (32) \\ &= 40 && (33) \end{aligned}$$

That is, about 40 tasks per client, for an arbitrary number of clients $n$. Of course, the client machine should reserve some power for consuming the data after filtering, so it should not run anywhere close to 40 such tasks.

Similarly, on the server, if we ignore $r$, Equation 6 tells us that

$$m \leq (\alpha^s S^s)\frac{t}{\mu} \tag{34}$$

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:

$$\begin{aligned} m &\leq \frac{\alpha^s S^s}{\mu} t && (35) \\ &= \frac{\alpha^c S^c}{\mu} \frac{\alpha^s}{\alpha^c} \frac{S^s}{S^c} t && (36) \\ &= \frac{1}{0.236\,\text{sec}} (0.75) \frac{S^s}{S^c} (10\,\text{sec}) && (37) \\ &= 31.7 \frac{S^s}{S^c} && (38) \end{aligned}$$

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 msec sample task as well as two other possibilities. The plot shows ratios $S^s/S^c$ reaching up to 200; large ratios can occur if, for example, the server is a large parallel computer or cluster, and the clients are simple palm-top devices.

In Figure 6 we show the constraints on $m$, in the agent case. This graph plots the two constraints from Table 2, as $d$ varies. The actual constraint is the minimum of the two curves. For lower $F'F$, the server's

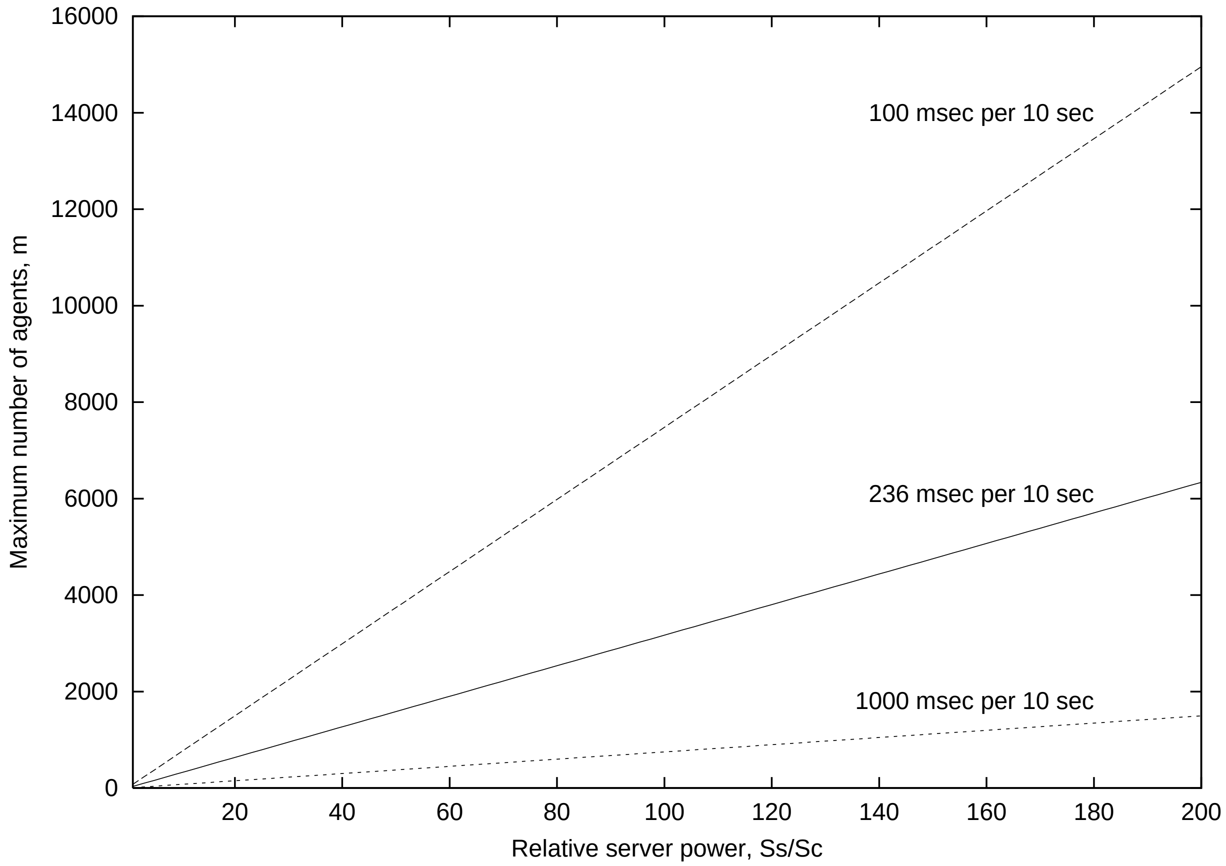


Figure 5: The number of agents that can effectively be supported, as the server power grows relative to the client's power. We show three curves, representing different possible computations; 236 msec represents our image-processing sample application.

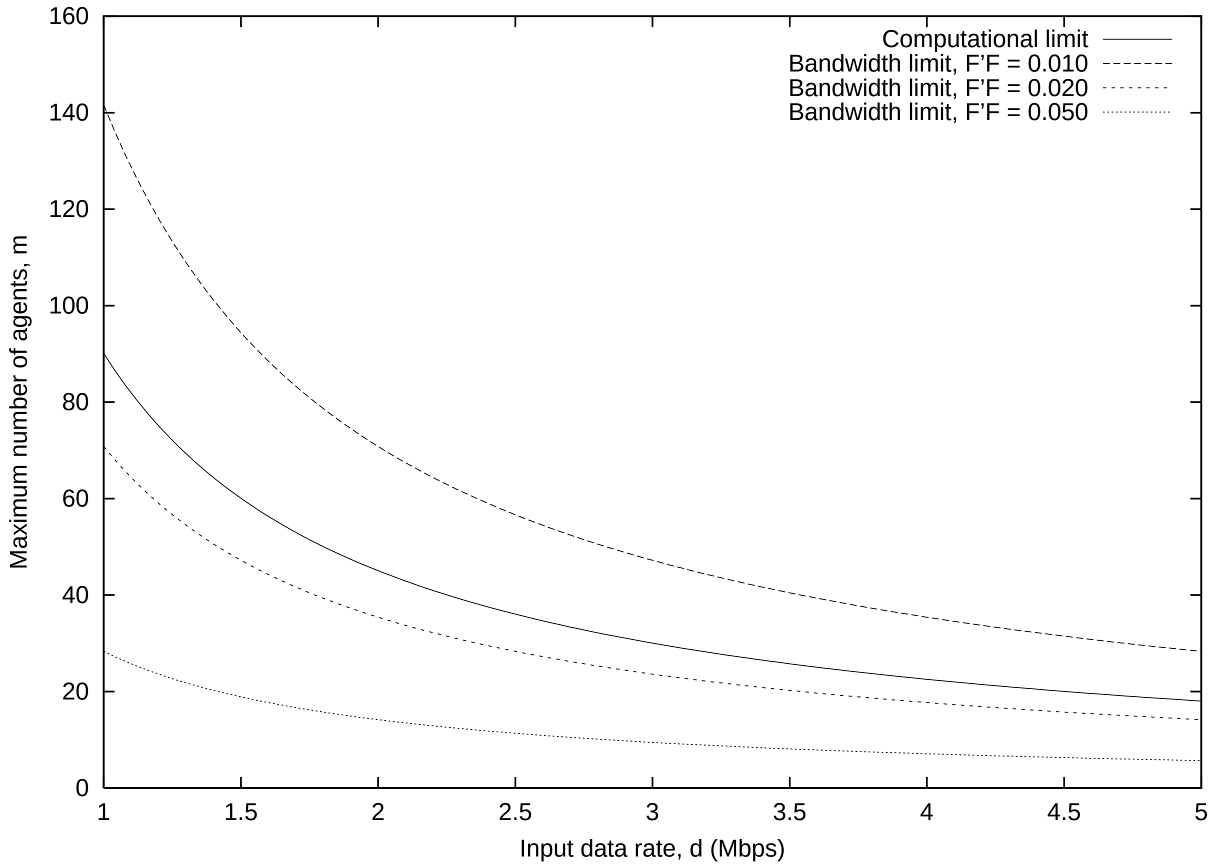


Figure 6: The maximum number of agents $m$ we can support, given the constraints in Table 2. Here $B = 2$ Mbps, $r = 0$, $\beta_a = 0.708$, $t = 10$ seconds, and $\mu$ is proportional to $D$, as described in the text.

computation is the tighter constraint; for higher $F'F$, the wireless network bandwidth limits us more. We use our earlier measurement of $\mu/\alpha^s S^s = 111$ msec (the edge-detection program running on a 308 KB image on one of our servers). Of course, in nearly any application $\mu$ will vary with $D$ (and thus with $d$ and $t$); for the purposes of this illustrative graph we assume that $t = 10$ and that the computation is linear. In other words, we imagine that $\mu$ may behave as follows.

$$\frac{\mu}{\alpha^s S^s} = 111\, msec \times \frac{D}{10\,\mathrm{sec} \times 1\,\mathrm{Mbps}} \tag{39}$$

In Figure 7 we look at similar results when we vary $r$ (the previous graph assumed $r = 0$). In Section 3.3 we measured

$$\frac{C_{init}}{\alpha^s S^s} = 40\,\mathrm{msec} \tag{40}$$

and in Section 3.1 we measured

$$\frac{\mu}{\alpha^s S^s} = 111\, msec \tag{41}$$

and for a fixed $t = 10$ seconds, the computational constraint from Equation 6 is

$$m \leq \left(\frac{\alpha^s S^s}{\mu} - \frac{rC_{init}}{\mu}\right)t \tag{42}$$

$$= \left(\frac{1}{111\,\mathrm{msec}} - r\frac{40\, msec}{111\,\mathrm{msec}}\right)(10\,\mathrm{sec}) \tag{43}$$

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. Above a certain

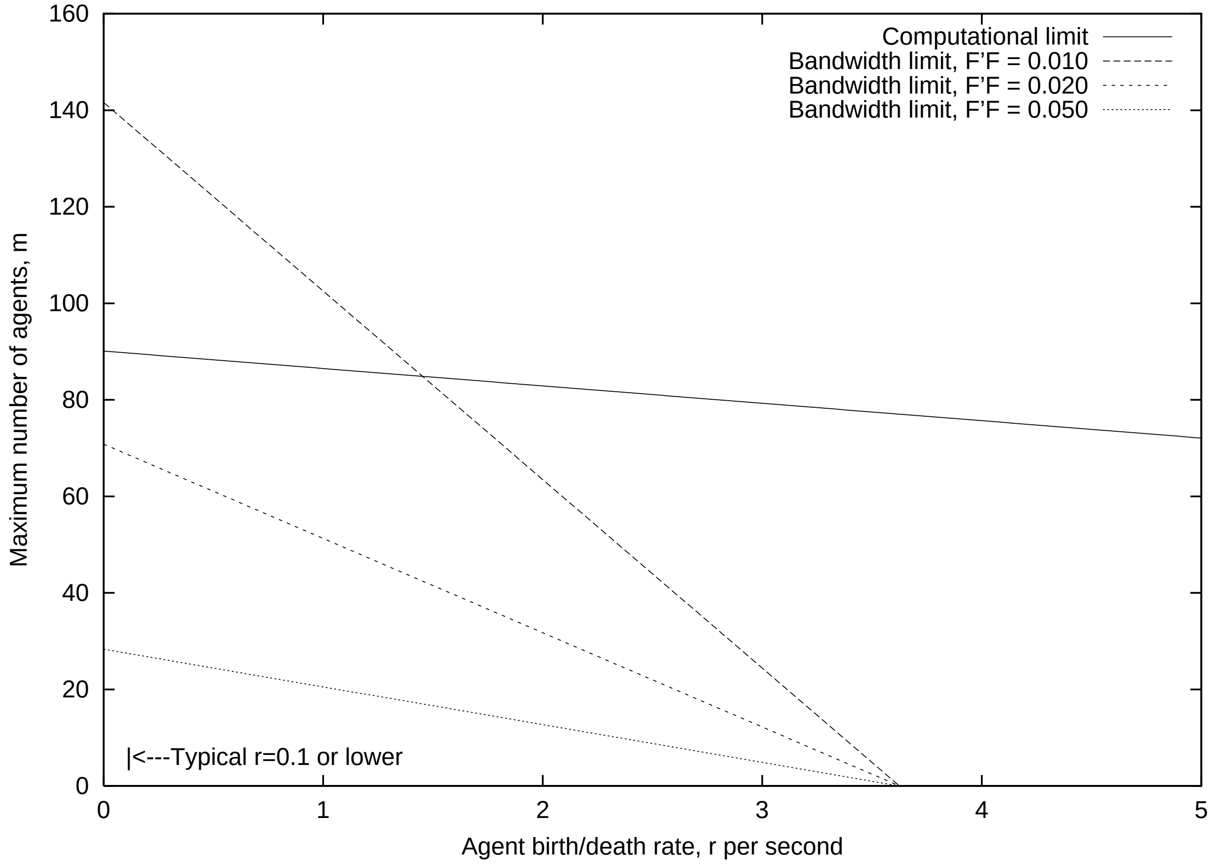


Figure 7: The maximum number of agents $m$ we can support, given the constraints in Table 1, as we vary $r$. Here $B = 2$ Mbps, $d = 1$ Mbps, $\beta_a = 0.708$, $t = 10$ seconds, $C_{init}/(\alpha^s S^s) = 40$ msec, and $\mu/(\alpha^s S^s) = 111$ msec.

point the traffic induced by the jumping agents $(rK)$ consumes the available bandwidth $B_a$, leaving nothing for agents to transmit their data. With a chunk size of $t = 10$, we think it highly unlikely that $r > 1$, and more likely $r << 1$.[6]

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 8 we vary the filtering ratio, since it clearly has a big impact on the bandwidth required by the agent approach. For low filtering ratios, the agent approach needs less bandwidth than the broadcast approach.

In Figure 9, however, we examine a case where $d > B$, and thus the broadcast approach cannot work. Note that the bandwidth required by the broadcast approach is $d/\beta_b$, and appears in both graphs slightly bigger than $d$.

## 5 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 [Kin90]. In addition, there has been some similar modeling work specifically for *mobile-agent* systems.

Straber and Schwehm [SS97] 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

[6] We have heard of some tests at Lockheed-Martin in which $r = 0.1$ at the peak.

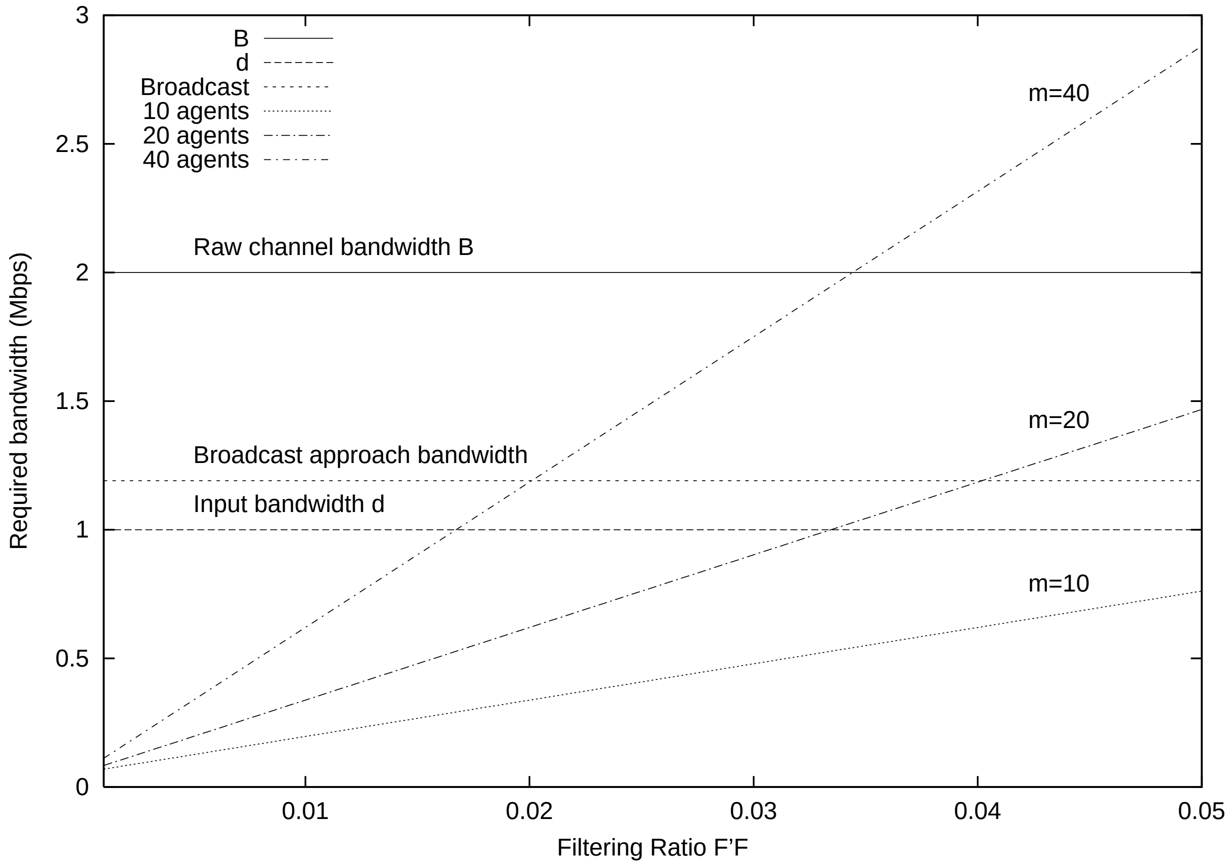


Figure 8: The bandwidth requirements for agent and broadcast approaches. Here $d = 1$ Mbps, $B = 2$ Mbps, $r = 0.1$, $K = 50$ KB, $\beta_a = 0.708$, and $\beta_b = 0.840$.

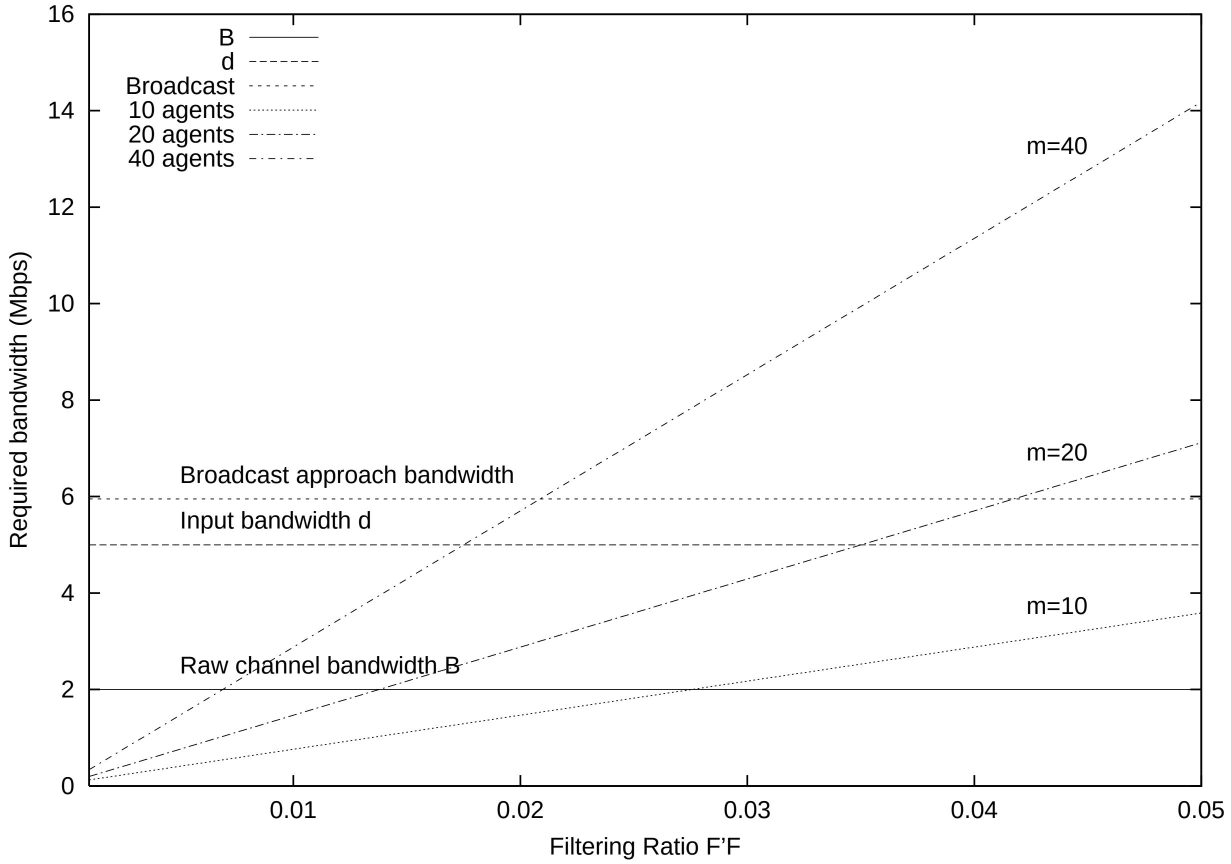


Figure 9: The bandwidth requirements for agent and broadcast approaches. The only difference from the previous graph is that $d = 5$ Mbps (and thus the broadcast approach is impossible). Unchanged are $B = 2$ Mbps, $r = 0.1$, $K = 50$ KB, $\beta_a = 0.708$, and $\beta_b = 0.840$.

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 [KP98] 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 [Pic98, FPV98] 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. [PRS99] use 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 Straber and Schwehm [SS97], 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 [SHG99], 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 [BP99], which examines the use of mobile agents for message delivery in ad-hoc wireless networks. Recent experimental work includes [SDSL99], which compares different strategies for accessing a Web database, and [GCKR00], 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 efficent compromise approach [IV96].

# 6 Summary and future work

Inspection of the above equations shows that with small filtering ratios ($F'F$), or small numbers of agents, a mobile-agent scheme can get by with less bandwidth, or slower (i.e., cheaper or lighter) clients. Our analysis reinforces the importance of the engineering challenge to keep $\alpha^s$ and $\beta_a$ large, that is, to reduce the overhead of mobile-agent computation and communication.

To further develop this performance analysis and to be able to use it predictively in real applications, we need to better understand the following:

- How variable is the input data stream, in its flow rate? In other words, how much buffering would be necessary in the server, and in the clients?
- How many different agent/task types are there in typical applications?
- 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 some of these limitations:

- The broadcast case assumes that nobody misses any transmissions, or that they do not care if they miss it, so there are no retransmissions.
- Both cases ignore the client processing consumed by the end application.
- We consider only one application scenario here. While it 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, and Niranjan Suri.

## References

[BP99] S. Bandyopadhyay and Krishna 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*, pages 69–73, Seattle, Washington, August 1999. ACM Press.

[CHK97] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile agents: Are they a good idea? In Jan Vitek and Christian Tschudin, editors, *Mobile Object Systems: Towards the Programmable Internet*, number 1222 in Lecture Notes in Computer Science, pages 25–47. Springer-Verlag, 1997.

[FPV98] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

[GCKR00] Robert S. Gray, George Cybenko, David Kotz, and Daniela Rus. Mobile agents: Motivations and state of the art. In Jeffrey Bradshaw, editor, *Handbook of Agent Technology.* AAAI/MIT Press, 2000. To appear. Draft available as Technical Report TR2000-365, Department of Computer Science, Dartmouth College.

[IV96] T. Imielinski and S. Viswanathan. Wireless publishing: Issues and solutions. In Tomasz Imielinski and Henry F. Korth, editors, *Mobile Computing*, chapter 11, pages 299–329. Kluwer Academic Publishers, 1996.

[Kin90] P. J. B. King. *Computer and Communication System Performance Modeling.* Prentice Hall, New York, 1990.

[KP98] Axel Küpper and Anothony S. Park. Stationary vs. mobile user agents in future mobile telecommunications networks. In *Proceedings of the Second International Workshop on Mobile Agents (MA '98)*, volume 1477 of *Lecture Notes in Computer Science*, Stuttgart, Germany, September 1998. Springer-Verlag.

[Pic98] Gian Pietro Picco. *Understanding, Evaluating, Formalizing and Exploiting Code Mobility.* PhD thesis, Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy, 1998.

[PRS99] A. Puliafito, S. Riccobene, and M. Scarpa. An analytical comparison of the client-server, remote evalutaion 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.

[SDSL99] George Samaras, Marios Dikaiakos, Constantine Spyrou, and Andreas 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)*, pages 50–64, Palm Springs, California, October 1999. IEEE Computer Society Press.

[SHG99] Tammo Spalink, John H. Hartman, and Garth 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)*, pages 42–49, Palm Springs, California, October 1999. IEEE Computer Society Press.

[SS97] Markus Straber and Markus Schwehm. A performance model for mobile agent systems. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, volume 2, pages 1132–1140, Las Vegas, June 1997.