Optimal on-line algorithms for an electronic commerce money distribution system

Hiroshi Kawazoe *, Tetsuo Shibuya †, and Takeshi Tokuyama ‡

*Industry e-Application Sales, e-business Solutions, IBM Japan,
19-21, Hakozaki-cho, Nihonbashi, Chuo-ku, Tokyo 103-8510, Japan.
† IBM Research Division, Tokyo Research Laboratory,
1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan.
‡ Graduate School of Information Sciences, Tohoku University,
2-1-1, Katahira, Aoba-ku, Sendai, Miyagi 980-8577, Japan.

E-mails: * kawazoe@jp.ibm.com, † tshibuya@trl.ibm.co.jp, ‡ tokuyama@dais.is.tohoku.ac.jp

keywords: on-line algorithm, algorithm analysis, electronic commerce, money distribution system, PayPerClick

Abstract

We consider the money distribution problem for a micro-payment scheme using a distributed server system; in particular, for an automatic charging scheme named *PayPerClick* that allows Internet users to view Web pages for which access charges are levied without tedious payment procedures. A major bottleneck in the scheme is the network traffic caused by the distribution of electronic money to many different servers. We propose a simple on-line algorithm for distributing electronic money to servers so that the network traffic is minimized. The algorithm achieves the optimal online competitive ratio. We also consider a *weighted* version, for which we give an asymptotically optimal online algorithm within a constant factor.

1 Introduction

In electronic commerce, the development of efficient schemes for charging and distributing of electronic money is a crucial issue [12]. One possible method of Web shopping is for users to make actual payments by sending credit card information for each purchase. This method is sometimes tedious and time-consuming. For example, when a user wants to see Web pages containing visual information such as artistic photographs, videos, and news, in cases where a charge is made for access, he or she should ideally be able to pay automatically without any interruption.

Although there are several cryptographic technologies such as electronic wallet [7] and offline electronic cash [6, 4, 11], they need somewhat heavy protocols in order to be applied to a micropayment scheme. One simple possible method (an online certification method) is for the user to send electronic cash to the shop, and for the shop to ask the user's payment server (say, a bank) to certify that the user's balance is sufficient, and transfer the amount to the shop's account in the shop's payment server. However, since the certification and money transfer process need to be highly secured [3, 12], they create a lot of network transactions; and therefore, we need a technology for reducing the number of certification and money transfer messages in the network. The MiniPay system [8] uses an online strategy, and reduces the number of transactions by allowing a small risk for the shop that a user may make a purchase without having enough money in his or her account. However, it is desirable to avoid this risk if possible.

A practical solution is to use trusted payment servers, and process most transactions within each server. Each user's real account is on the user's main payment server (called the home server), but the purchase and payment are basically done on the shop's payment server, which we call a guest server from the viewpoint of the user. Each server is the shared payment server of multiple shops, as well as a home server multiple users. Figure 1 shows a chart of our model. A payment at a guest server can be completed without any communication with the home server if the guest server has a certain amount of money that has been transferred from the user's home server. The distribution of the user's money should be dynamically controlled by using some clever strategy.

An example of such a payment system is the PayPerClick [5] system [1] for charging users to view Web pages containing visual information. In this system, a payment is made automatically when the user clicks on a link to a charged-for Web page and agrees to pay the fee. The Web pages are managed by using special servers called PPC servers, with each Web page assigned to one of the servers. A typical implementation of this charging scheme is a prepaid system, in which a user prepays a fixed number of units (say, n units) of electronic money. We can also consider a postpaid scenario in which the user receives acknowledgment each time he/she uses n units.

In order to avoid heavy transactions on a server, we use multiple servers, and hence the bottleneck of this solution is the network communication caused by the dynamic distribution of electronic money among the servers. In this paper, we loosen the bottleneck by using a simple on-line money distribution algorithm.

The problem is somewhat similar to the famous K-server problem [13, 10], where we must choose how K mobile servers will serve a sequence of requests occured in the metric space. The goal is to minimize the transportation cost. In our case, we can regard electronic cash as mobile

[1] Under development at IBM Japan

servers, and a network of the payment servers as the metric space. The difference is that the amount of electronic cash is reduced by a unit (or by the amount of the payment) when a payment is made, and the amount of cash sent in a transaction does not affect the transportation cost. There have been many studies of online algorithms for distributed systems and the Internet [1, 2, 9]. However, to the authors' knowledge, this is a new application of online algorithms.

We show that if the offline best strategy requires k message sets, our online algorithm creates at most $k \log(n/k) + 2k$ message sets, where each set consists of at most four messages on the network. Since we can prove that $k \log(n/k) - O(k)$ message sets are necessary for any online strategy, our algorithm attains an asymptotically optimal competitive ratio (in fact, this is a little different from the usual competitive ratio analysis [13], since the ratio $\log(n/k)$ depends on k.) We also generalize our algorithm for the case in which the total cost of traffic is considered instead of number of messages, and show analytically that the algorithm attains an asymptotically optimal performance if a constant factor is ignored.

2 Money distribution problem in the PayPerClick scheme

Although our methods can be applied to the general problem of distributing money on a distributed payment server system, we use the PayPerClick problem to illustrate the use of our algorithms. Suppose a user (say, Alice) initially buys n units of electronic cash, which are considered to be prepaid money for viewing "charged-for" Web pages. A charge is made each time an item is clicked in charged-for Web pages. The prices may depend on the Web items; however, we assume that it is one unit for each click, since the general case can be reduced to this case.

We consider the following network model. We have a set of servers $\{p_0, p_1, \ldots, p_M\}$, where p_0 is Alice's home server, and the others are guest servers. Let $balance(p_i)$ be the amount of Alice's electronic cash kept at server p_i. Each server has a set of charged-for Web pages. If Alice clicks an item in a Web page assigned to p_i, $balance(p_i)$ is decreased by 1. However, if $balance(p_i) = 0$, the server p_i sends a *remittance request*, and obtains cash from other servers. We assume that the remittance request is always sent to the home server p_0, and that cash is sent through p_0.

A very naive implementation of a PayPerClick scheme is to keep all electronic money at the home server, and to send a unit of cash to a guest server every time a click occurs at that server. This creates n network messages, which would cause a serious network traffic problem. Another naive implementation is to distribute money evenly to all guest servers. In this case, however, M may be very large, whereas a typical user accesses only a small number of servers.

On the other hand, if we know in advance where all the n clicks will occur, and let n_i clicks occur on p_i, the optimal strategy is to send n_i units to p_i the first time the home server receives a remittance request from p_i. If there are k guest servers sending remittance requests, this strategy requires only $2k$ messages (k remittance requests and k money-transfer messages). Unfortunately, we cannot foresee future clicks, and hence we need an on-line strategy.

One possible online strategy is to send a fixed number of units (say, t units) of cash to each guest server when it sends a remittance request to the home server. This strategy seems to reduce the amount of traffic to approximately n/t messages. However, it may happen that we send out all n units from the home server at the first n/t clicks, and need to collect cash from guest servers to

make the next remittance; this reduces the performance if t is large. Therefore, we need a better strategy.

More generally, we consider the weighted-cost version in which the minimized target is the total cost of the traffic, where the cost of sending a message between p_0 and p_i is c_i. The case in which $c_i = 1$ for each i is the above basic problem, which we call the *unit-cost* problem.

3 A lower bound

In this section, we give a lower bound for any online strategy. This lower bound is for the weighted-cost version of the PayPerClick problem.

Suppose that the servers $p_1, \ldots, p_k$ are clicked during the consumption of n units of money. We assume that any cash flow from p_i to p_j is via the home server p_0. Let c_i be the cost of communicating between p_i and p_0. Without loss of generality, we assume that $c_1 = c_{max}$ is the maximum communication cost. Let $C = c_1 + ... + c_k$.

We begin by stating the following well-known fact:

Lemma 3.1 *If* $0 \leq x \leq 0.5$, $1 - x \geq 2^{-2x}$.

Theorem 3.2 *Any online algorithm incurs* $\Omega(C \log\{(n+k)/k\})$ *costs for the remittance messages in the worst case. Moreover, if* $c_{max} < C/2$, *it incurs* $(C/2)\log\{(n+k)/k\}$ *costs, where the base of* log *is* 2.

Proof: First, we assume that $c_{max} \leq C/2$. Suppose that $n_i(t)$ units of money are given to p_i just after the t-th remittance request (which we describe below) is made. Let $n(t) = \sum_{i=1}^{k} n_i(t)$, $f_i(t) = (n_i(t) + 1)/c_i$, and $f(t) = \min_i f_i(t)$.

The strategy of an adversary is to always click the i-th server with the minimum f_i. Then, after at least $n_i(t) + 1$ clicks, the adversary forces a remittance message to be sent from p_i, and we charge c_i, since we need to bring some money to p_i. Note that the algorithm may create some network communication during the period, but we ignore it (i.e., we do not charge) unless it is a remittance message from p_i. Hence, we can assume that $cost(t+1) \geq cost(t) + c_i$, where $cost(t)$ is the cost up to the t-th remittance request. We permit any relocation of money to be made free of charge before starting the next stage.

The amount of money is decreased by $n_i(t) + 1$. Since $f(t) \leq (n(t) + k)/C$, $n_i(t) + 1 \leq [c_i(n(t)+k)/C]$. Thus, $n(t+1) \geq (1-c_i/C)n(t) - c_i k/C$. Therefore, $n(t+1)+k \geq (1-c_i/C)(n(t)+k)$. From the lemma, $1 - c_i/C \geq 2^{-2c_i/C}$. Hence, if the balance is reduced to zero after T iterations, $(n+k)2^{-2cost(T)/C} < k$. Therefore, $2cost(T)/C > \log\{(n+k)/k\}$, and $cost(T) > (C/2)\log\{(n+k)/k\}$.

Next, if $3C/4 \geq c_1 > C/2$, suppose that the amount of money is decreased by a factor of Y by clicks on p_1. Then, $\log_4 Y$ remittance requests must be sent by p_1, and hence a cost of $c_1 \log_4 Y > (C/2)\log_4 X$ is incurred. On the other hand, the amount of money must be decreased by a factor of $X \geq (n+k)/kY$ by clicks on other servers, and the cost must be at least $(C/2)\log Y$. Hence, the total cost must be $\Omega(C \log\{(n+k)/k\})$.

Finally, if $c_1 > 3C/4$, we change the adversary's strategy to one such that it clicks p_1 if $n_1(t) < n(t)/2$, and otherwise clicks p_i that minimizes $f(t)$. If $n_1(t) \geq n(t)/2$, $f(t) \leq (n(t)+k)/(C/2) \leq 2(n(t)+k)/C$, $n_i(t)+1 \leq 2c_i(n(t)+k)/C$, and $n(t+1)+k \geq (1-2c_i/C)(n(t)+k)$. Since $c_i < C/4$, we can use the lemma and $1 - 2c_i/C \geq 2^{-4c_i/C}$. Hence, if we spend $cost_{small}(T)$ on such clicks, and the amount of money is decreased by a factor of X, $4cost_{small}(T)/C > \log X$. On the other hand, if $n_1(t) < n(t)/2$, $n(t+1)+k > (n(t)+k)/2$. Hence, if we spend a total of $cost_{large}(T)$ on clicking p_1 and the amount of money is decreased by a factor of Y, then $cost_{large}(T) > C_1 \log_2 Y > (3C/4)\log_2 Y$. Since $XY \geq (n+k)/k$, we have the $\Omega(C\log\{(n+k)/k\})$ bound.

□

Since the optimal offline algorithm requires C remittance messages, the above result gives an $\Omega(\log\{n/k\})$ lower bound for the competitive ratio. Setting $c_i = 1$ for all i, we have the following:

Corollary 3.3 *For any online algorithm, $(k/2)\log\{(n+k)/k\}$ messages are necessary for the unit-cost problem.*

The constant factor of the above corollary will be slightly improved later.

The above lower bound uses an adaptive adversary, which decides what action to take with knowledge of the current status of the money distribution. For analyzing a randomized online algorithm, an oblivious adversary, which selects an action without knowing the current status, is usually considered [13]. We can show that our asymptotic lower bound can be attained by using such an oblivious adversary.

Proposition 3.4 *Any online algorithm requires $\Omega(C\log((n+k)/k))$ messages against an oblivious adversary.*

Proof: For readability, we assume that $C_{max} < C/2$. Suppose that the adversary clicks on servers $p_1, \ldots, p_k$. Our oblivious sequence of clicks is as follows:

1: while $n > 0$;
2: Select $i \in \{1, 2, .., k\}$ with probability c_i/C
3: Perform $\lfloor 2c_i n/C \rfloor + 1$ clicks on p_i
4: $n = n - \lfloor 2c_i n/C \rfloor - 1$
5: end while

For each while loop, a remittance message occurs with probability 0.5, and hence we can perform a similar analysis to that used for the above theorem. □

4 An almost optimal algorithm for the unit-cost case

In practice, the unit-cost problem is important, and we give an asymptotically optimal algorithm for it in this section. Our simple online strategy for the unit-cost problem is basically as follows (see also Figure 2):

Algorithm. Initially, all n units are kept at the home server. When an empty server p_i is clicked, p_i sends a remittance message to the home server p_0, and p_0 selects the server p_j with the

maximum balance and asks it to send half (actually, $\lceil balance(p_j)/2 \rceil$) of its cash to p_0; then, p_0 transfers the cash to p_i.

There is a deception in the above strategy: The home server cannot determine the exact current balance of other servers without sending additional messages. For the time being, we ignore the above defect, and assume that the home server knows the exact balance. We later remove this restriction by making a minor modification to the algorithm. Without loss of generality, we assume that the home server p_0 also sends remittance information to itself if a click occurs when it is empty. Since each remittance request is accompanied by a *message set* (to be precise, it is named *regular message set*) consisting of at most two pairs of money-transfer messages between the home server and guest servers, we count only the number of remittance requests.

Lemma 4.1 *If the optimal offline algorithm requires k messages, the above algorithm requires at most $k \log(n/k) + 2k$ remittance requests.*

Proof: Since the optimal offline algorithm requires k messages, n clicks occur on k servers.

We claim that the worst scenario for the above algorithm is that the clicking is done by the LCB (least current balance) adversary, which clicks a server with the least current balance. Note that this scenario is the same as that given to show the lower bound in the previous section. If this claim holds, it suffices to consider the performance of our algorithm for LCB. Indeed, each of k servers is clicked (since they are initially empty) in the first k clicks, and the largest balance at this stage is at most $\lfloor 2n/k \rfloor$, there are at most k remittance requests until the largest balance is reduced by half, and hence $k(2 + \log(n/k))$ remittance requests suffice.

The claim is proved by induction on n. If $n = 1$, the claim is trivial, since we need only one remittance message if the click is made on an empty server, and no remittance message otherwise. We assume that the claim holds if the total amount of cash is $n-1$. Let Y be the sequence obtained by arranging the balance values of servers in a non-increasing order. $|Y|$ is the sum of the sequence, which is the total amount of cash. Let $f^s(Y)$ be the sequence obtained from Y by our algorithm after s clicks which occurs in the LCB order. Suppose that there is a stronger adversary OPT than LCB, and let $g^s(Y)$ be the sequence after s clicks have been made by OPT. From the induction hypothesis, $g(X) = f(X)$ for $|X| \leq n-1$. Therefore, $g^s(Y) = f^{s-1}(g(Y))$, since $g(Y)$ has only $n-1$ units of cash.

Let p_i be the server with the minimum balance in Y. If $balance(p_i) \geq 1$, suppose the optimal algorithm clicks on p_j, where $balance(p_j) > balance(p_i)$. So far, no remittance message has been created. From the induction hypothesis, we see that the next click will occur on p_i in both adversaries. Hence, we see that g^2Y can also be obtained by clicking on p_j in $f(Y)$. However, from the induction hypothesis, the strongest adversary on $f(Y)$ is LCB, which clicks p_i. This is a contradiction.

If $balance(p_i) = 0$, we consider a server p_s with the maximum balance in Y. LCB creates a remittance message and resets $balance(p_i)$ to $\lceil balance(p_s)/2 \rceil - 1$ and $balance(p_s)$ to $\lfloor balance(p_s)/2 \rfloor$. Suppose OPT clicks on p_j, where $balance(p_j) \geq 1$. Then, the next click on $g(Y)$ must be made on an empty server (without loss of generality, p_i) by OPT and create a remittance message. Indeed, $g^2(Y)$ is obtained by clicking on p_j in $f(Y)$ if $j \neq s$ and on either p_i or p_s if $j = s$. This contradicts the fact that the strongest adversary on $f(Y)$ is LCB. □

We now modify our algorithm so that knowledge of the exact current balance of servers is not necessary. Instead of the real balance, we use the temporary balance $temp(p_j)$, which is the balance of server p_j when the last message was sent between p_j and p_0. Precisely speaking, if the last message was a transfer of x units of money from p_0 to p_j, then $temp(p_j) = x$; otherwise, the last message should be the transfer of $\lceil balance(p_j) \rceil$ units of money from p_j to p_0, and $temp(p_j) = \lfloor balance(p_j) \rfloor$, where balance information is sent to p_0 together with the money. In the modified algorithm, if the remittance request is issued, the server with the maximum temporary balance is chosen, and half of the (real) balance at the server is sent back to the home server.

Theorem 4.2 *The modified algorithm creates at most* $k \log(n/k) + 2k$ *remittance requests.*

Proof: Let p_j be the server with the maximum temporary balance. Let $temp(p_j) - balance(p_j) = h$ when the remittance request at a server p_i is created. If $balance(p_j) = 0$, our algorithm issues another remittance request so that the request at p_i is covered. We perform the following shuffling of the order of clicks. The h clicks on p_j are postponed until after the remittance request, and then $\lfloor h/2 \rfloor$ clicks are made on p_j and $\lceil h/2 \rceil$ clicks are made on p_i. Note that, if $balance(p_j) = 0$, this makes an additional remittance request at p_i. Notice that both the configuration of the balances at the end and the number of remittance requests are not modified by this shuffling order of clicks. Moreover, the original algorithm using the real balances and the modified algorithm using the temporary algorithm perform in the very same way if the clicks are made in the rearranged order. Thus we conclude that the performance of the modified algorithm is not worse than the worse-case performance of the first algorithm. □

From Corollary 3.3, the above bound is asymptotically tight, if we ignore a factor of 2. Moreover, we can show a stronger lower bound. We count the number of message sets: A *regular message set* consists of a remittance request from the current server to the home server, a money transfer request from the home server to a guest server, a money transfer from the guest server to the home server, and a money transfer from the home server to the current server. If either the current server is the home server or the home server has money to transfer to the current server directly, the message set consists of fewer messages. An *irregular message set* consists of a pair of messages between two servers (through the home server) which transfers money when no remittance request is issued. This can be also created as an additional money transfer message set to a regular message set just after a remittance request is issued.

Lemma 4.3 *Any online algorithm needs* $k \log(n/k) - k$ *message sets.*

Proof: Suppose that after a completion of a remittance and a money transfer, the balance of the poorest server is less than $N/2k$, where N is the current total balance. We call this situation "skew"; otherwise, we call this situation "smooth". The initial situation is skew, since the home server has all the money. Suppose that we have a smooth situation after z remittance requests, and that the current balance is N. Without loss of generality, we can assume that $N \geq k$. After a skew situation, the next message set is issued against the LCB strategy before the total balance is reduced to $(1 - 1/2k)N$. Therefore, $z > k \log\{n/N\}$, by the same argument as Lemma 4.1. Suppose that, in a smooth situation, the server p_i has N_i units. Because of the smoothness, $N_i > N/2k$.

We will prove later that the optimal algorithm need not use an irregular message set, and hence we first consider the case where the algorithm generates only regular message sets.

After the initial smooth situation, the adversary first empties the k-th server, and then clicks on the poorer of the two servers involved in the last money transfer (either the sender or the receiver). This decreases some N_i to at least $\lfloor N_i/2 \rfloor$. Then, it is easy to see that $\sum_{i=0}^{k-1} \lfloor \log N_i \rfloor \geq (k-1)\log(N/2k)$ remittance requests are necessary until all the money is used up. Hence, the total number of remittance requests must be at least $k\log\{n/N\} + k\log(N/2k)$, which is $k\log(n/k) - k$.

Next, we prove that irregular message sets will not improve the performance. Suppose that the optimal algorithm needs at most $M < k\log(n/k) - k$ message sets in total. The optimal adversary strategy is not bothered until the first irregular message set is issued. We assume that it is issued after L regular message sets. Let $D(init)$ be the money distribution after the L regular message sets. Suppose that the adversary's next move for $D(init)$ against an algorithm without irregular messages is to click m times the server p_i to empty it if no irregular message set is sent. The next move of our adversary is to do the same (click m times on p_i) on the current distribution; however, if the irregular message sets reduces the balance of p_i before that, we just empty it.

Consider the time period between the L-th regular message set and the end of the next move of the adversary. The irregular message set sending x units of money from p_i to p_j is denoted by $send(i, j; x)$. If both $send(i, j; x)$ and $send(j, l; y)$ are executed and $x \geq y$, we can replace them by the pair of $send(i, j; x-y)$ and $send(i, l; y)$ to give the exactly same money distribution. Therefore, we can assume that each server is either a source server (only sending money) or a sink server (only receiving money) during the period.

If p_i is a sink server and $send(j, i; x)$ has been done, it is not emptied with the next move (m clicks on p_i). However, we can consider another algorithm which first waits at $D(init)$, then issues $send(j, i; x)$ (as a regular message set) when p_i is emptied, and then does the rest irregular message sets. This algorithm attains the same money distribution as the original one, has M message sets, and its first $L+1$ message sets are regular.

If p_i is a source server, it is emptied, and suppose y units are sent from the server p_j by the algorithm. Suppose that p_j originally had n_j units of money and has received $send(s(q), j; x(q))$ $(q = 1, 2, .., h)$. Let u be the smallest index such that $n_j + \sum_{q=1}^{u} x(q) \geq y$. If $n_j \geq y$, we set $u = 0$.

Then, we can design another algorithm which first waits at $D(init)$ until p_i becomes empty, and then receives n_j units from p_j (if $n_j = 0$, receives $send(s(1), i; x(q))$ from $p_{s(1)}$) as a regular message set, and then executes the rest of irregular message sets except that (1) messages sent from p_i are removed and (2) $send(s(q), j; x(q))$ is changed to $send(s(q), i; x(q))$ for each of $q \leq u$. It is easy to observe that the balance of each server of the resulting distribution is not more than that of the current distribution. Our adversary reduces the current distribution to this distribution by applying some clicks.

Hence, we can design an algorithm in which (1) the number of message sets is at most M and (2) the first $L+1$ message sets are regular. The next move of our adversary is as if it competes with this new algorithm.

Therefore, by induction, there exists an algorithm with at most M messages without an irregular message set, which is a contradiction. □

The number of messages depends on each message set; it is two if the current server is the home

server or the home server sends its own money to the current server, and four otherwise. If we use a simplified model which neglects the above difference, the above lower bound implies that our algorithm is optimal up to an $O(k)$ additive term.

5 Weighted version

In this section, we consider a weighted version of the PayPerClick problem, in which each remittance message between p_0 and p_i has transportation cost c_i and the goal is to minimize the total cost of the traffic.

Without loss of generality, we assume that servers p_i $(1 \leq i \leq k)$ are the k servers that will be clicked. Let c_{min} and c_{max} be the smallest and largest costs among c_i $(1 \leq i \leq k)$. We can easily see that the total cost of the optimal offline algorithm for this problem is $C = \sum_{1 \leq j \leq k} c_i$.

We can directly apply the online algorithm given in the last section for this weighted problem, and it is observed that it incurs a cost of $c_{max} \cdot (2k \log(n/k) + 4k)$ at most.

We propose the following algorithm with a better performance ratio:

Algorithm. Initially, all n units of money are kept on the home server p_0, and let f be n/c_{max}. Repeat the following procedure until n units of money have been used up:

1. Let p_i be the clicked server. If p_i has any money, decrease $balance(p_i)$ by 1 unit of money and go to step 5. Otherwise, p_i sends a remittance request to the home server p_0.

2. If the home server p_0 has more money than $c_i f/2$, send $\lceil c_i f/2 \rceil$ units of money from p_0 to p_i, decrease $balance(p_i)$ by 1 unit of money, and go to step 5.

3. Let $\{p_{s_1}, p_{s_2}, \ldots p_{s_{k-1}}\}$ be the list of $\{p_j | j \neq i\}$ sorted descendingly according to the value of $balance(p_j)/c_j$. Let l be the smallest l such that $c_i \leq \sum_{1 \leq j \leq l} c_{s_j}$ or $\lceil c_i f/2 \rceil \leq balance(p_0) + \sum_{1 \leq j \leq l} \lceil balance(p_{s_j})/2 \rceil$. If no such l exists, let l be $k-1$. Send $\lceil balance(p_{s_j})/2 \rceil$ units of money from p_{s_j} to the home server p_0 for each $1 \leq j \leq l$.

4. If the home server p_0 has at least $c_i f/2$ units of money, send $\lceil c_i f/2 \rceil$ units of money from p_0 to p_i. Otherwise, send $balance(p_0)$ units of money from p_0 to p_i. Decrease $balance(p_i)$ by 1 unit of money, and go to step 5.

5. Let f' be $\max\{\max_{1 \leq i \leq k}\{balance(p_i)/c_i\}, balance(p_0)/c_{max}\}$. If $f/2 \geq f'$, let f be $f/2$.

Figure 3 shows the behavior of this algorithm. The figure shows the remittance behavior in the case that neither the clicked server nor the home server has any money. Notice that this algorithm, like the first unit-cost algorithm in the last section, uses the value of $balance(p_i)$ which cannot be known to the actual algorithm. We will modify it later. Before showing the total cost of this algorithm, we obtain the following two lemmas:

Lemma 5.1 *In the algorithm, f must be updated after spending $3C + 5c_{max}$ message cost.*

Proof: By definition of f, $balance(p_i) \leq c_i f$ for each $1 \leq i \leq k$, and $balance(p_0) \leq c_{max} f$ at the first step of any iteration of the algorithm. The remittance message cost is charged only in

steps 2, 3, and 4 of an iteration procedure. Consider a period which does not include any part of an iteration procedure updating f. If a server is chosen to send money to the home server in step 3, its balance never exceeds $c_i f/2$ afterwards. Moreover, since the list of p_j in step 3 is in the descending order according to $balance(p_j)/c_j$, once a server whose balance is not larger than $c_i f/2$ was chosen to send money to the home server in step 3 of the algorithm, f must be updated in step 5. Therefore, no such server can be chosen in step 3 during the period. This means that each server is chosen in step 3 at most once during the period. Thus the total remittance message cost in step 3 must be smaller than C during the period.

Next, let us consider the message cost incurred in steps 2 and 4. Let multiset $\{p_{t_1}, p_{t_2}, \ldots, p_{t_m}\}$ be the servers clicked when they are empty during this period. Note that the servers that are clicked s times will appear s times in the set, and we distinguish them in the following discussion. The total cost incurred in steps 2 and 4 during this period is thus $\sum_{1 \le i \le m} c_{t_i}$.

Without loss of generality, we assume that $\{p_{t_1}, p_{t_2}, \ldots, p_{t_r}\}$ are the servers to which $\lceil c_{t_i} f/2 \rceil$ units of money are sent from the home server, and that $\{p_{t_{r+1}}, p_{t_{r+2}}, \ldots, p_{t_m}\}$ are the servers to which fewer than $\lceil c_{t_i} f/2 \rceil$ units of money are sent from the home server.

First, we consider the servers p_{t_i} $(i \le r)$. The cost spent in steps 2 and 4 for them is $C' = \sum_{1 \le j \le r} c_{t_j}$. During this period, we send remittance messages from the home server to servers $\{p_{t_1}, p_{t_2}, \ldots, p_{t_r}\}$ (in steps 2 and 4) for $\sum_{1 \le j \le r} \lceil c_{t_j} f/2 \rceil$ units of money, which is not smaller than $C'f/2$.

At the beginning of this period, the home server has at most $c_{max} f$ units of money. Thus we must send at least $C'f/2 - c_{max} f$ units of money in total from servers to the home server in step 3 when the clicked server is p_{t_i} $(i \le r)$. Each of the servers has only $c_i f$ units of money at most, so the total remittance message cost in step 3 when the clicked server is p_{t_i} $(i \le r)$ is at least $C'/2 - c_{max}$.

Next, consider the servers p_{t_i} $(i > r)$. The cost at steps 2 and 4 for them is $C'' = \sum_{r < j \le m} c_{t_j}$. There are two cases (see step 3 in the algorithm): Case 1. $c_{t_i} \le \sum_{1 \le j \le l} c_{s_j}$ in step 3. Case 2. $l = k - 1$ in step 3. However, if $l = k - 1$, it is observed that f must be updated in step 5, which means case 2 never occurs in the period. In case 1, the message cost incurred at step3 is at least that at steps 2 and 4 (*i.e.*, C'').

Thus, total message cost incurred at step 3 is at least $C'/2 - c_{max} + C''$. According to the upper bound of the message cost incurred at step 3, it must be smaller than C: $C'/2 - c_{max} + C'' < C$. The message cost incurred at step 2 and 4 is $C' + C''$, which is smaller than $2C + 2c_{max}$ because $C' + C'' \le C' + 2C'' = 2 \cdot (C'/2 - c_{max} + C'') + 2c_{max} < 2C + 2c_{max}$.

Hence the message cost during a period which does not include any part of an iteration updating f is smaller than $2C + 2c_{max} + C = 3C + 2c_{max}$.

Consider the message cost incurred during an iteration. The cost incurred at step 2 or 4 is at most c_{max}, and that at step 3 is at most $2c_{max}$. Thus the cost is at most $3c_{max}$ in total. Finally, we conclude that f must be updated after spending at least $3C + 2c_{max} + 3c_{max} = 3C + 5c_{max}$ message cost. □

Corollary 5.2 *The cost of the algorithm is at most* $(3C + 5c_{max}) \log n$.

We sharpen the above result by using the following lemma:

Lemma 5.3 *The algorithm spends at most* $10C - 4c_{max} + 2Ck/n \log(C/c_{max})$ *message cost until* f *becomes smaller than* n/C.

Proof: Let t_i be the total number of clicks until the click causing the ith remittance request, *i.e.*, the t_ith click is the ith click on empty servers. Let u_j be the number of remittance requests before the jth update of f. Let $v_j = t_{u_j}$ and $f_i = 2^{-i}n/c_{max}$ be the value of f after the ith update of f. Let p_{w_i} be the server clicked at the t_ith click, and let C_i be $\sum_{1 \le j \le u_i} c_{w_j}$.

In the beginning, the home server has all the money, and $f = n/c_{max}$. According to step 2 of the algorithm, if the sum of the costs of the clicked (empty) servers exceeds c_{max}, the balance of the home server is reduced to below $n/2$, and f must be updated to $f_1 = n/(2c_{max})$ and $C_1 < c_{max} + c_{u_1} \le 2c_{max}$.

We can observe that, after the v_ith click, $\sum_{i=1}^{k} balance(p_i) \le C_i f_i$ and $balance(p_0) \le c_{max} f_i$. A requesting server p_i requests $\lceil c_i f_i/2 \rceil$ units of money, and a requested server p_j sends $\lceil balance(p_j)/2 \rceil$ units of money. Thus if the sum of the costs of the clicked empty servers after the v_ith click exceeds $(\sum_{i=1}^{k} \lceil balance(p_i) \rceil + balance(p_0))/(f_i/2) \le (C_i f_i/2 + k/2 + c_{max} f_i)/(f_i/2) = C_i + 2c_{max} + k/f_i$, f must be updated.

Thus $C_{i+1} \le C_i + (C_i + 2c_{max} + k/f_i) + 2c_{w_{u_{i+1}}} \le 2C_i + 3c_{max} + k/f_i$. We can see that $C_i \le 2^{i-1}(C_1 + 3c_{max} + kc_{max}(i-1)/n) - 3c_{max} < (5 + k(i-1)/n) \cdot 2^{i-1} c_{max} - 3c_{max}$, because $f_i = 2^{i-1} f_1 = 2^{-i} n/c_{max}$.

Let $l = \lceil \log_2(C/c_{max}) \rceil$. It is obvious that $l \ge 1$ and $f_l \le n/C$. Then $C_l < (5 + k(l-1)/n) \cdot 2^{l-1} c_{max} - 3c_{max} < (5 + k/n \log(C/c_{max}))C - 3c_{max}$. Thus, the total cost until the v_lth click in steps 2 and 4 of the algorithm is smaller than $(5 + k/n \log(C/c_{max}))C - 3c_{max}$.

After the v_ith click, the total cost in step 3 until the v_{i+1}th click is at most $C_{i-1} + 2c_{max}$. Thus the total cost in step 3 until the v_lth click is smaller than $\sum_{i \le l}(C_{i-1} + 2c_{max})$. It is smaller than $(5 + k/n \log(C/c_{max}))C - c_{max}$ because $\sum_{i \le l}(C_{i-1} + 2c_{max}) < \sum_{i \le l}((5 + k(i-2)/n) \cdot 2^{i-2} c_{max} - 3c_{max} + 2c_{max}) < (5 + k(l-2)/n) \cdot 2^{l-1} - c_{max} \le (5 + k/n \log(C/c_{max}))C - c_{max}$.

Hence we conclude that f becomes smaller than n/C after a remittance message cost at least $10C - 4c_{max} + 2Ck/n \log(C/c_{max})$ has been incurred. □

Theorem 5.4 *The algorithm incurs a remittance message cost that is at most* $(3C + 5c_{max}) \log(c_{max} n/C) + 10C - 4c_{max} + 2Ck/n \log(C/c_{max})$.

Proof: From Lemma 5.3, after spending $10C - 4c_{max} + 2Ck/n \log(C/c_{max})$, f is updated $\log(c_{max} n/C)$ times and f is reduced to $1/c_{max}$, which means that the system is empty. The result follows from Lemma 5.1. □

Since $balance(p_i)$ cannot be known to the actual algorithm, we must modify this algorithm to use $temp(p_i)$ instead of $balance(p_i)$. We obtain the following theorem for the modified algorithm:

Theorem 5.5 *The modified algorithm incurs a remittance message cost that is at most* $(3C + 5c_{max}) \log(c_{max} n/C) + 10C - 4c_{max} + 2Ck/n \log(C/c_{max})$.

Proof: We can shuffle the order of the clicks to let $temp(p_i) = balance(p_i)$ without changing the total remittance message cost, as in the case of the unit-cost algorithm. Let $h_j = temp(p_j) - balance(p_j)$ when the remittance request at a server p_i is created. We postpone h_j clicks on p_j until

after the remittance request. Furthermore, we let the postponed $\lceil h_j \rceil$ clicks be made on p_i and $\lfloor h_j \rfloor$ clicks be made on p_j. For this click sequence, the results of the original algorithm and the modified algorithm are the same, and the remittance cost is same as in the case of the original click sequence by the modified algorithm. This shows that the worst-case cost by the modified algorithm is not worse than the original algorithm. □

Hence, for the weighted problem, the above algorithm has a better bound than the algorithm for the unweighted problem discussed in the last section: the former algorithm costs $O(C \log(c_{max} n/C))$ and achieves asymptotic optimality within a constant factor, while the latter costs $O(kc_{max} \log(n/k))$.

6 Concluding remarks

We have presented optimal on-line algorithms for the problem of distributing money on a payment server system in electronic commerce. Although we have given lower bounds, they hold only for a simple server-network model using connections between a home server controlling the whole process and guest servers. A more sophisticated distributed network model might make it possible to create a system with a better performance.

In practice, the algorithms should be mixed with some other heuristic rules. For example, we usually bound the number k of active guest servers, and money remaining at guest servers should be returned to the home server at a fixed interval (for example, every quarter year). Also, users should be advised to supplement their cash before spending all n units, in order to improve the performance.

A preliminary demonstration system for the PayPerClick is available [5] (unfortunately, only in a Japanese version) but it is a virtual prototype which does not support any real payment, and the current prototype does not have a money distribution scheme. Hence we are going to implement our algorithms. We will also apply electronic signature and other security techniques so that users can claim compensation in the event of malicious actions by servers.

References

[1] B. Awerbuch, Y. Bartal, and A. Fiat, Distributed Paging for General Networks, *Journal of Algorithms* **28** (1998), pp. 67-104.

[2] B. Awerbuch, Y. Bartal, and A. Fiat, Heat and Dump: Randomized Competitive Distributed Paging, *Proc. 34th IEEE FOCS* (1993), pp. 22-31.

[3] N. Asokan, E. van Herreweghen, M. Steiner, Towards a Framework for Handling Disputes in Payment Systems, Research Report RZ 2996, IBM Research, March 1998

[4] S. Brands, Untraceable Off-line Cash in Wallet with Observers, *Proc. Crypto '93* (1993), 302-318.

[5] PayPerClick Demonstration site (in Japanese), http://ppc.fiesta.or.jp/

[6] D. Chaum, A. Fiat, and M. Naor, Untraceable Electronic Cash, *Proc. Crypto'90, LNCS 403,* (1990), pp. 319-327.

[7] S. Even, O. Goldreich, and Y. Yacobi, Electronic Wallet, *Proc. Crypto 83* (1983) pp. 383-386.

[8] A. Herzburg and H. Yochai, MiniPay: Charging per Click on the Web, *Proc. 6th WWW Conference* (1997), pp. 239-253.

[9] S. Irani, Page Replacement with Multi-size Pages and Applications to Web Cashing, *Proc. 29th ACM STOC* (1997), pp. 701-710.

[10] S. Irani and A. Karlin, Online Computation, In *Approximation Algorithms for NP-hard problems, ed. D. Hochbaum*, PWS Publishing Company, 1997.

[11] T. Okamoto, An Efficient Divisible Electronic Cash Scheme, *Proc. of Crypto'95, LNCS 963* (1995), pp. 438-451.

[12] P. Wayner, *Digital Cash*, 2nd ed., Academic Press, 1997.

[13] R. Motowani and P. Raghavan, *Randomized Algorithms*, Cambridge Press, 1995.

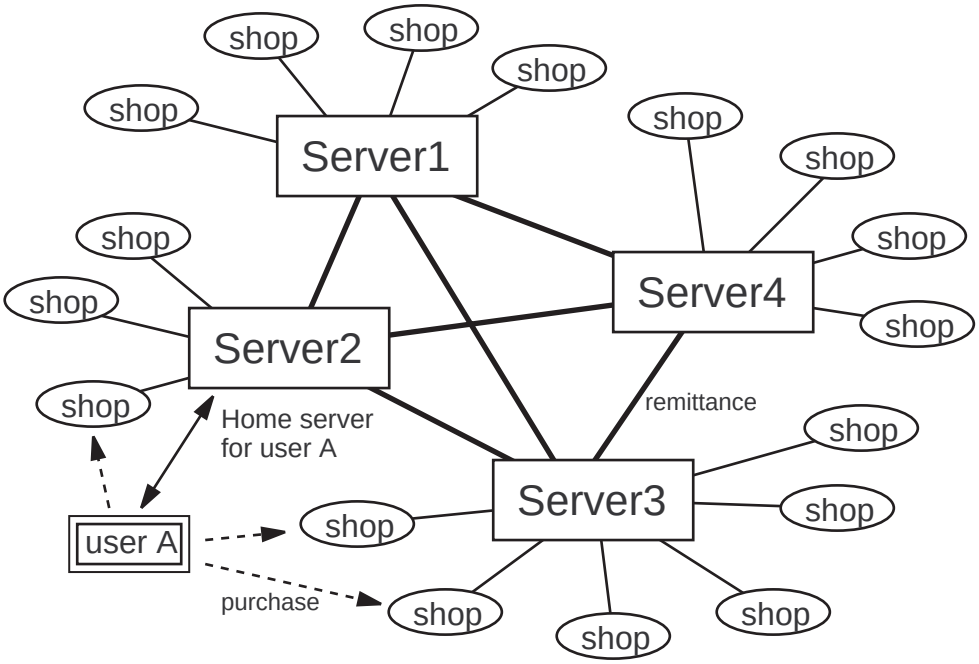

Figure 1: Muti-server model for a micro-payment system

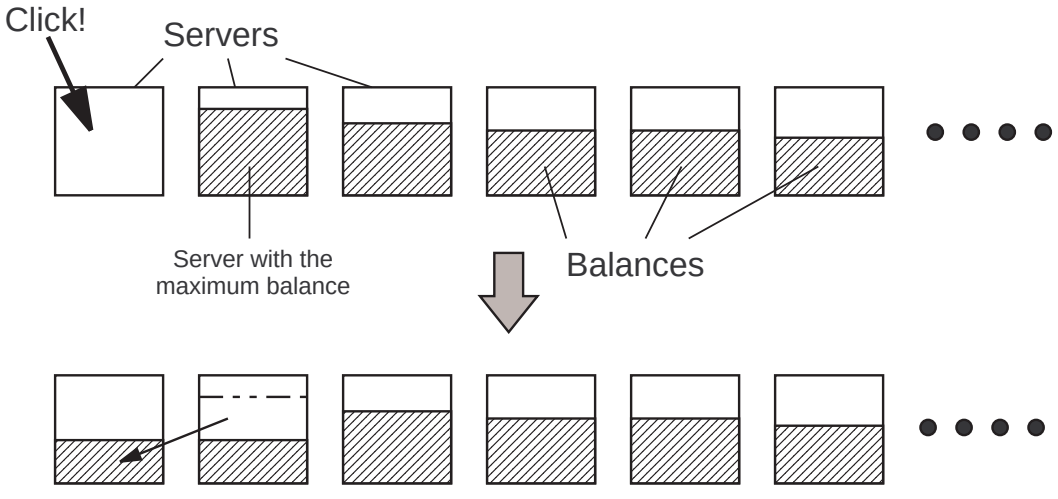

Figure 2: Behavior of the basic algorithm for the unit-cost case

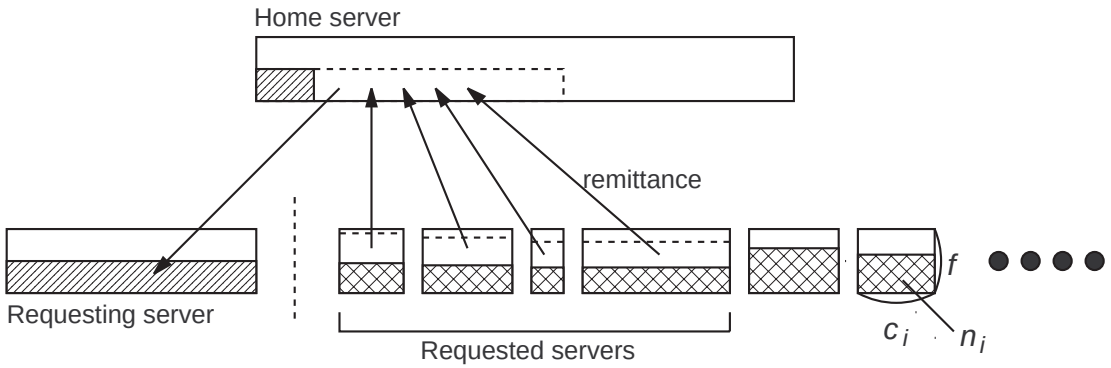

Figure 3: Behavior of the basic algorithm for the weighted cost case