University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS) | Department of Computer & Information Science

January 2002

On-line payment method for small amount transactions using hierarchical prepaid cards

Toru Egashira

Jonathan M. Smith
University of Pennsylvania, jms@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_reports

Recommended Citation
Egashira, Toru and Smith, Jonathan M., "On-line payment method for small amount transactions using hierarchical prepaid cards" (2002). *Technical Reports (CIS)*. Paper 154.
http://repository.upenn.edu/cis_reports/154

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-02-28.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_reports/154
For more information, please contact repository@pobox.upenn.edu.

On-line payment method for small amount transactions using hierarchical prepaid cards

Abstract
Traditional payment methods such as credit cards are not suitable for small amount payment because the transaction may cost certain commission comparable to the small amount payment. When we buy a cheap item at real shop, we pay by cash. But, how about online shopping where you cannot pay by cash? There are a ton of contents online having potential value to sell but not as expensive as credit card payment fits - such as multimedia clips. How an online shopper can buy them? A widely used solution is aggregation: online merchants bundle several (sometimes unnecessary) items together to multiply the price. Another solution is having payment method suitable for small amount, aka *micropayment*. There has been many research-level micropayment schemes developed, using cryptographic schemes to eliminate the need to access central server for authorization purposes - what researchers claim costly operation. Such schemes with "off-line" verification enable banks or other participants to trace back to a crook that cheats by double spending. However, such schemes cannot stop double spending itself and thus a crook can run away before they become aware of the cheat[?].

Comments
University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-02-28.

This technical report is available at ScholarlyCommons: http://repository.upenn.edu/cis_reports/154

On-line payment method for small amount transactions using hierarchical prepaid cards (Extended Abstract)

Tōru Egashira, and Jonathan M. Smith
egashira@bq.jp.nec.com, jms@central.cis.upenn.edu

Abstract

Abstract follows...

Keywords

Keyword One, Keyword Two.

1. Introduction

Traditional payment methods such as credit cards are not suitable for small amount payment because the transaction may cost certain commission comparable to the small amount payment. When we buy a cheap item at real shop, we pay by cash. But, how about online shopping where you cannot pay by cash? There are a ton of contents online having potential value to sell but not as expensive as credit card payment fits — such as multimedia clips. How an online shopper can buy them? A widely used solution is aggregation: online merchants bundle several (sometimes unnecessary) items together to multiply the price. Another solution is having payment method suitable for small amount, aka *micropayment*. There has been many research-level micropayment schemes developed, using cryptographic schemes to eliminate the need to access central server for authorization purposes — what researchers claim costly operation. Such schemes with "off-line" verification enable banks or other participants to trace back to a crook that cheats by double spending. However, such schemes cannot stop double spending itself and thus a crook can run away before they become aware of the cheat [?].

This paper proposes a payment scheme with low-cost on-line authorization. The scheme bases on a prepaid-card system: a customer first buy a (virtual) card from a card issuer, then she pay merchants for items or services using the card up to the amount of its face value. With hierarchically structured prepaid cards and wallet software installed on every customer terminal, we relieve reliability requirements for servers and thus lower server operation costs that affect authorization cost for each transaction.

2. Cost of transactions

We first define a simplified model of a payment-card-based transaction system. There are three roles: customer, issuer, and merchant in the model (Figure 1). In the model, there are many customers and merchants, but we assume there is only a single issuer in the world for simplicity. A customer provides the information of his/her card, which the issuer issued, to pay a merchant for merchandise. The merchant then forwards the card data to the issuer and asks for the transaction to be authorized. If successfully authorized, the merchant provides merchandise as the customer requests. The issuer issues payment cards to customers and receive funds from them. As we think of prepaid-card-based scheme, the issuer receives money as a result of selling prepaid cards to customers. The issuer maintains a database that has records of accounts each corresponds to an issued card, and makes decisions whether transactions should be approved based on data in these records. An authorized transaction results in the issuer's fund transfer to the merchant. The issuer might not pay the amount of transaction fully: it deducts some amount as a commission aka "merchandise discount." The

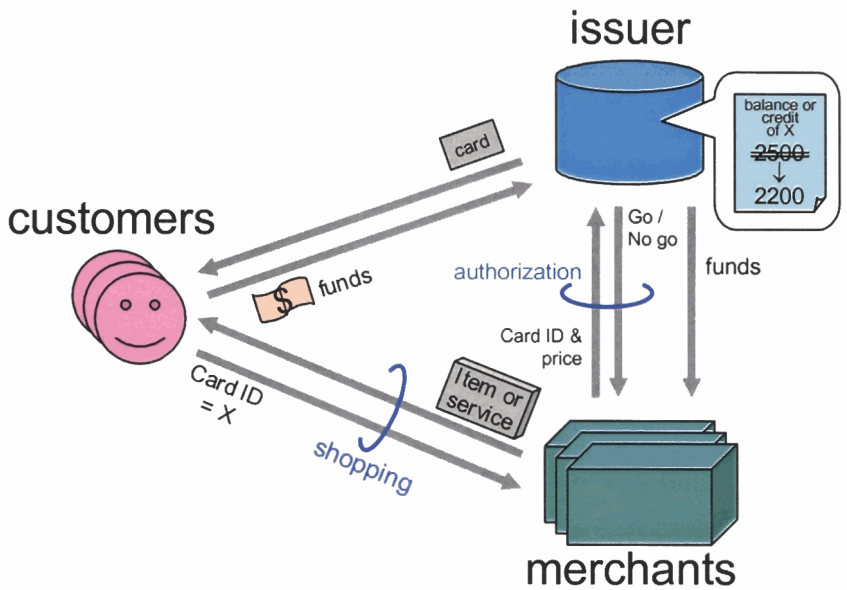

Figure 1. A simple model of a transaction system.

commission is the issuer's main source of revenue unless there are other big sources such as annual membership fee and installment interest as is the case of credit cards. But, from the viewpoint of merchants, it is the transaction cost they must pay.

The transaction cost involves many factors. Some of them are technical matters like this study deals with, and others are not. Some kind of business incurs more overhead costs than others: bank is an example. Banks are under bank law enforcement that needs wide disclosure of their operations and other regulations to meet. Though such regulations let them reliable, they force certain amount of overhead cost on banks. If you are going to start up a business that manages individual accounts for general public to add and withdraw their funds, you must be a bank and those regulations are yours. But if you just deal with prepaid cards, which are not bound to an individual and people cannot add funds, you don't have to be a bank (at least in Japan), and you can be competitive in terms of operation costs. This is the first reason we propose a prepaid-card-based scheme (other reasons are technical, which will be discussed later on).

We deal with the cost of authorization system operation. The system must work reliably 24/7, or merchants may lose business chances, which is also critical for issuer business. The higher the reliability requirement, the higher the system operation costs generally — we have to overcome this formula to establish a small-amount payment system.

3. Reducing the cost

We now introduce two techniques to reduce the reliability requirement. The first technique is account caching by customers instead of by servers. The second technique is the use of hierarchical cards to increase customers' chances to have spare cards. These techniques are not mutually exclusive but rather complementary and we can use both in a system.

3.1. Customer caching

We assume that issuer's authorization system consists of more than one server clusters each has its own account database. Our clusters each manages a mutually exclusive part of total accounts space, and it might consist only a single host. The slicing of accounts will naturally distribute authorization-processing load among these clusters.

Now we are going to allow one of these clusters to go down while (almost) all transactions are correctly processed. We put "almost" because in certain condition we may allow fraud transactions (but the risk will be small as shown later). To allow a cluster to be down, we must have a spare account record for every account on somewhere other than the cluster that has the original. These spare records are normally managed on another cluster. We, however, let customers have these spare and don't let clusters permanently have spare records. This will reduce the number of accesses to clusters and account databases: when we update a "master" account record (e.g. due to the deduction from the balance), the change has to be propagated to the spare record within some short period of time, and that involves accesses to another cluster and its DB if we store spares into a neighbor cluster, but we don't.

When one of the clusters failed and a customer is going to make a payment to a merchant using a card of which the failed cluster has the master record, what happens? The merchant now realizes the cluster it is asking for authorization is down, and then asks the secondary cluster for help. The secondary cluster checks if the cluster is really failed; creates a spare record for the card, taking the balance data as the customer provides; and authorizes later payments that uses the corresponding card based on the record until the failed cluster recovers. When the failed cluster recovers, it updates the master record according to the spare record. The secondary cluster does not have to be a dedicated one but another cluster primarily for its own slice of accounts space. But, some attention is needed for such "mutual help" system — if we assign a cluster *B* to be a secondary cluster for all records in a cluster *A*, chain reaction may happen: when *A* go down because of heavy processing load, all the load will next knock out *B*. Thus, spare records for a cluster should be distributed among all other clusters, in a manner that the spare record for card *A1* is on cluster *B*, *A2* is on cluster *C*, and so on, for the additional processing load on surviving clusters to be distributed. Of course, all merchants must know the identical mapping rule that specifies which cluster has spares for which cards.

As described above, we depend on customers to have copies of card records. But how can they do that? We assume every customer has wallet software that helps customer's processing of transactions and it will cache account records also. A customer makes every online payment through the wallet software, so it knows how much the customer has spent and amount left. Suppose a debit card system that the customer cache is applied the same way, things don't work well: as the account is a checking account of a bank, we can modify the account record by issuing checks, ATM withdrawal, wire transfer, and so on; thus the customer cache linked to only debit card usage become stale easily. Prepaid card system, however, we neither can add funds nor withdraw through back doors, and thus customer cache sounds. This is the second reason we propose a prepaid-card-based scheme.

We must be aware of a possible fraud using a false wallet: a wallet that reports its card has full balance regardless of the actual balance. When the user of the wallet make a payment while the cluster for the card is down, a spare record will be created as the wallet reports — full balance, as nobody can check the report at that time. Until the failed cluster revives, the user can enjoy shopping as much as the full balance of the card. At later time when the cluster revives, it will know there was a fraud, and the issuer may cancel all transactions falsely made. Though it can stop merchant to stop shipping items, we cannot take back already transmitted multimedia contents.

There is a measure for the fraud, but far from perfection. We should make wallets to always report the current balance of a card it is going to use, and the cluster should check if it is correct. When the balance does not match, the cluster rejects the transaction, and possibly suspends further payment from the card. That will reject frauds using false wallets that ALWAYS report incorrect balance — we cannot stop fraud if the user can know which cluster is down and which card is enabled for the fraud.

Though we cannot stop the fraud completely, we can make the risk and incentive of fraud to be small enough using the next technique: hierarchical cards. More explanation will come later.

3.2. Hierarchical cards

Imagine you are thirsty and buying a bottle of drink at a vending machine; you put coins in the slot, but one of them are rejected. What do you do? You just put another coin if you have. In the same manner, we ask a customer for another prepaid card when a card cannot be processed due to cluster failure. Of course, the customer might have only the card that failed and we are not going to force customers to have two or more cards. There is a way to increase the possibility for a customer to possess two or more cards: selling cards as a bunch, not individually. So is a hierarchical card, which is sold to and handled by a person as an indivisible unit, but it has actual cards inside. We call a card that people recognize as a "major card," and actual cards inside a major card as "minor cards."

To bridge the gap between what people recognize and what the system processes, we again assume customers use wallet software to make payment. A user first registers her major cards to the wallet; then it handles them as bunches of minor cards; it makes payment using one or more of these minor cards; and when a merchant asks you for another card, it provides another minor card if it has.

Though the probability for a customer to possess another minor card will increase with this technique, it is less than one. If there are no another cards, the transaction will not be authorized. What probability a transaction fails? There is another concern: how much cluster accesses increase? As we put a bunch of major cards into a major card, the value of a minor card must of course be smaller than the major. So, a customer has to use two or more minor cards to make a payment — that involves two or more clusters to be accessed. Next section provides analysis of these concerns.

4. Analysis of hierarchical cards

5. Conclusions and Future Work

In this paper, we have presented the architecture of a node inquiry function (NIF) that helps on-line service providers to place front-ends strategically. The strategic placement plan that the NIF provides enables a service provider to easily obtain the best possible front-end distribution that maximizes investment efficiency. The node filter function in the NIF helps service providers to find front-end nodes that can run their front-ends: it accepts predicates and finds those heterogeneous front-end nodes that satisfy each predicate. The placement planner function in the NIF makes placement plans, which enhances the service quality regardless of how many front-ends are finally deployed. The algorithm combines brute-force and greedy approaches to calculate placement plans in polynomial time.

For future directions of the NIF, it will need more input from on-line service providers to make more efficient placement plans. A NIF-suggested placement plan, once generated, will soon become obsolete due to changes in demand and so on. A service provider thus may need to have a new plan suggested midway through the old plan. The current NIF, however, assumes that no front-ends have already been deployed. A future NIF should accept the provider's input of current deployment and take it into consideration.

The cost and demand matrices in the cost map are currently applied for every on-line service. To calculate more efficient placement plans, this information should reflect service-specific metrics and usage data, which only the service provider knows. A future NIF should provide cost maps for each on-line service and customize them according to the provider's feedback.

References

1. Author, "Paper," In Proc., Date.
2. A. B, and C. D, "Improved Foo Bar Solution," In *40th IEEE Symp. Baz of Comp. Sci.*, October 1999.