

NetBill: An Internet Commerce System Optimized for Network Delivered Services

Marvin Sirbu
Engineering and Public Policy Dept.

Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

J. D. Tygar
Computer Science Dept.

Abstract

Netbill is a business model, set of protocols, and software implementation for commerce in information goods and other network delivered services. It has very low transaction costs for micropayments (around 1¢ for a 10¢ item), protects the privacy of the transaction, and is highly scalable. Of special interest is our new certified delivery mechanism which delivers information goods if and only if the customer has paid for them. This paper discusses the design of the NetBill protocol and our World Wide Web (WWW) prototype implementation

Introduction

As the explosive growth of the Internet continues, more people rely on networks for timely information. However, since most information on the Internet today is free, intellectual property owners have little incentive to make valuable information accessible through the network. There are many potential providers who could sell information on the Internet and many potential customers for that information. What is missing is an electronic commerce mechanism that links the merchants and the customers.

NetBill is a business model, set of protocols, and software implementation allowing customers to pay owners and retailers of information. While NetBill will enable a market economy in information, we still expect that there will be an active exchange of free information.

The market for information

Porat and others have shown that information industries dominate the economy [1]. Estimates of the market for on-line information vary from \$10 billion to \$100 billion per year depending upon how the market is defined [2]. There are more than 15,000 databases accessible over networks. Vendors can distribute information products varying from complex software valued at thousands of dollars per copy, to journal pages or stock quotes valued at a few pennies each. A challenge for network-based electronic commerce is to keep transaction costs to a small fraction of the cost of the item. The desire to support *micropayments* worth only a few pennies each is a driving factor in the NetBill design.

A second challenge in the information marketplace is supporting *micromerchants*, who may be individuals who sell relatively small volumes of information. Merchants need a simple way of doing business with customers over networks, so that the costs of setting up accounting and billing procedures are minimal. A model for micromerchants is the French Minitel system, which provides 20,000 “kiosks” offering computer-based services to Minitel users. Many of these kiosks are provided by small entrepreneurs who enter the marketplace for little more than the cost of a PC and the labor to acquire or develop valuable information.

The purchase of goods over a network requires linking two transfers: the transfer of the goods from the merchant to the customer, and the transfer of money from the customer to the merchant. In the case of physical goods, a customer can order the goods and transfer money over the network, but the goods cannot be delivered over the network. Information goods have the special characteristic that both the delivery of the goods *and* the transfer of money can be accomplished on the same network. This allows for optimizations in the design of an electronic commerce system.

A NetBill scenario

Figure 1 shows NetBill’s model. A user, represented by a client computer, wishes to buy information from a merchant’s server. A NetBill server maintains accounts for both customers and merchants. These accounts are linked to conventional financial institutions. A NetBill transaction transfers the information goods from merchant to user, and debits the customer’s account and credits the merchant’s account for the value of the goods. When necessary, funds in a customer’s NetBill account can be replenished from a bank or credit card; similarly funds in a merchant’s NetBill account are made available by depositing them in the merchant’s bank account.

The transfer of an information good consists of delivering bits to the customer. This bit sequence may have any internal structure, for example, the results of a database search, a page of text, or a software program. Users may be charged on a per item basis, or by a

subscription allowing unlimited access, or by a number of other pricing models.

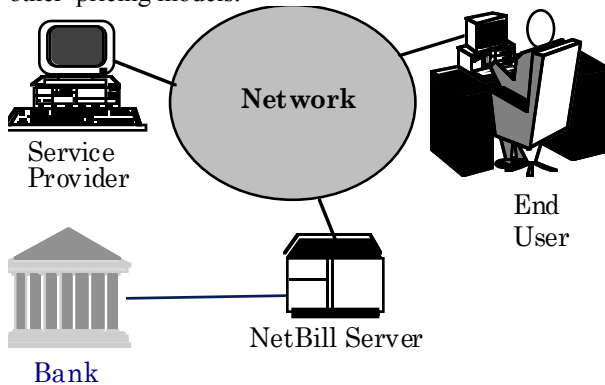


Figure 1

Once the customer receives the bits, there are no technical means to absolutely control what the customer does with them. For example, suppose an information provider wants to charge a different price for pages viewed on-line, versus printed pages. The merchant can provide customers with client software distinguishing viewing from printing, and which initiates a new billing transaction when the screen is printed. However, there are no technical means to prevent the user from tampering with that software once it is on her machine; a corrupt user who has only paid to view the bits could thus bypass the charge for printing. Merchants may still choose to distribute special software in the belief that tampering will be infrequent. Similarly, there is no technical means to prevent users from violating copyright by redistributing information [3].

NetBill design

There are a number of challenges to making electronic commerce systems feasible:

- *High transaction volumes at low cost.* If information is sold for a few pennies a page, then an electronic commerce system must handle very large transaction volumes at a marginal cost of a penny or less per transaction.
- *Authentication, privacy and security.* The Internet today provides no universally accepted means for authenticating users, protecting privacy, or providing security.
- *Account management and administration.* Users and merchants must be able to establish and monitor their accounts.

This paper describes the architecture of NetBill, a system designed to meet these goals. Our students and we have implemented three generations of NetBill prototypes. We hope to soon mount a trial in which various forms of information are sold to users using NetBill.

NetBill architecture

NetBill uses a single protocol that supports charging in a wide range of service interactions. NetBill provides transaction support through libraries integrated with

different client-server pairs. These libraries use a single transaction-oriented protocol for communication between client and server and NetBill; the normal communications model between client and server is unchanged. Clients and servers can continue to communicate using protocols optimized for the application — for example, video delivery or database queries — while the financial-related information is transmitted over protocols optimized for that purpose. This approach allows NetBill to work with information delivery mechanisms ranging from the WWW to FTP and MPEG-2 streams.

The client library — which we call the *checkbook* — and the server library — the *till* — have a well-defined API allowing easy integration with a range of applications. (Below we describe how we integrated these libraries with Mosaic clients and HTTP servers.) The libraries incorporate all security and payment protocols, relieving the client/server application developer from having to worry about these issues. All network communications between the checkbook and till are encrypted to protect against adversaries who eavesdrop or inject messages.

The NetBill transaction protocol

Before a customer begins a typical NetBill transaction, she will usually contact a server to locate information or a service of interest. For example, the customer may request a Table of Contents of a journal showing available articles available, and a list price associated with each article. The transaction begins when the customer requests a formal price quote for a product. This price may be different than the standard list price because, for example, the customer may be part of a site license group, and thus be entitled to a marginal price of zero [4]. Alternatively, the customer may be entitled to some form of volume discount, or perhaps there is a surcharge during the peak hour.

Requesting the price quote is easy. As we discuss below, in a WWW browser application we have built, a customer requests a price quote by simply clicking on a displayed article reference.

The customer's client application then indicates to the checkbook library that it would like a price quote from a particular merchant for a specified product. The checkbook library sends an authenticated request for a quote to the till library which forwards it to the merchant's application. (Figure 2, Step 1.)

The merchant then must invoke an algorithm to determine a price for the authenticated user [5]. He returns the digitally signed price quote through the till, to the checkbook (Step 2), and on to the customer's application. The customer's application then must make a purchase decision. The application can present the price quote to the customer or it can approve the purchase without prompting the customer. For example, the customer may specify that her client software accept any price quote below some threshold amount; this relieves her of the burden of assenting to every low-value price quotes via a dialog box.

Assume the customer's application accepts the price quote. The checkbook then sends (Step 3) a digitally signed purchase request to the merchant's till. The till then requests the information goods from the merchant's application and sends them to the customer's checkbook encrypted in a one-time key (Step 4), and computes a cryptographic checksum (such as MD5 [6]) on the encrypted message. As the checkbook receives the bits, it writes them to stable storage. When the transfer is complete, the checkbook computes its own cryptographic checksum on the encrypted goods and returns to the till a digitally signed message specifying the product identifier, the accepted price, the cryptographic checksum, and a timeout stamp: we refer to this information as the *electronic payment order (EPO)* (Step 5). Note that, at this point, the customer can not decrypt the goods; neither has the customer been charged.

Upon receipt of the EPO, the till checks its checksum against the one computed by the checkbook. If they do not match, then the goods can either be retransmitted, or the transaction aborted at this point. This step provides very high assurance that the encrypted goods were received without error.

If checksums match, the merchant's application creates a digitally signed invoice consisting of price quote, checksum, and the decryption key for the goods. The application sends both the EPO and the invoice to the NetBill server (Step 6).

The NetBill server verifies that the product identifiers, prices and checksums are all in agreement. If the customer has the necessary funds or credit in her account, the NetBill server debits the customer's account and credits the merchant's account, logs the transaction, and saves a copy of the decryption key. The NetBill server then returns to the merchant a digitally signed message containing an approval, or an error code indicating why the transaction failed (Step 7). The merchant's application forwards the NetBill server's reply and (if appropriate) the decryption key to the checkbook (Step 8).

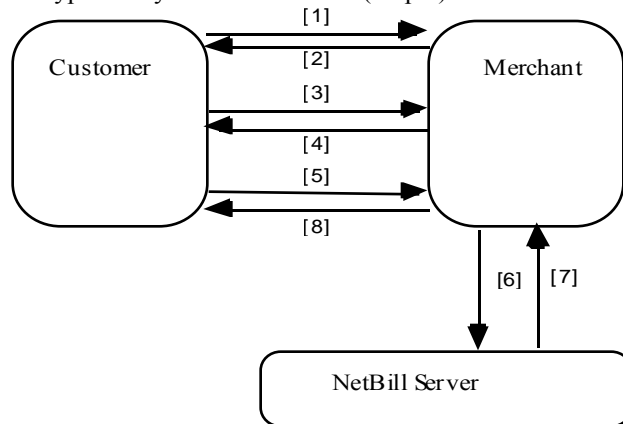


Figure 2- Transaction Protocol

Protocol Failure Analysis

The above description assumed that no failures occurred during the execution of the protocol. In reality, the

protocol must gracefully cope with network and host failures. One of our goals is to tightly link two events: charging the customer and delivering the goods. The customer should pay exactly when she receives the information goods.

The NetBill server is highly reliable and highly available. All transactions at the NetBill server are atomic: they either finish completely or not at all. NetBill is never in doubt about the status of a purchase. We cannot make similar assumptions about the reliability of the merchant's and customer's software; they must maintain a state consistent with the NetBill Server.

First, consider the protocol from the perspective of the customer's application. Up to step 5, when the customer application acknowledges receipt of the information goods, the customer application knows that no transaction has occurred. That is, the customer does not have access to the product and the merchant does not have the customer's money. Once the application sends the EPO, the customer is *committed* to the transaction and must be prepared to accept the purchase. If the customer's application does not receive a response from the merchant's application, then it is the responsibility of the customer's application to determine what happened: the customer's application can poll either the merchant application or the NetBill server to determine the status of the purchase request. If the merchant's application did not successfully forward the EPO to the NetBill server, then the EPO will have expired and the NetBill server will respond to the customer's application that the purchase has failed. Of course, the customer still does not have the one time key, so while the customer still has her money, she also does not have the goods. If, on the other hand, the transaction succeeded before communication failed, then the customer's application can find the status of the purchase and, if appropriate, the decryption key from either the merchant's application or the NetBill server (which has registered the key). If both are unreachable, the customer's application must continue to poll.

Now consider the protocol from the perspective of the merchant's application. Before it forwards the EPO and invoice to the NetBill server, the merchant's application knows that the transaction has not occurred. After it forwards the EPO and invoice, however, the merchant's application is *committed* to the transaction and must obtain the result from the NetBill. If the merchant's application does not receive a response from the NetBill server, the merchant's application must poll the NetBill server.

The protocol is much simpler for the NetBill server than for the other parties. The NetBill server is never in a state in which it depends on a response from another entity to determine the status of a transaction. Until the NetBill server receives the EPO and invoice from the merchant's application, it knows nothing about the purchase. Once it receives the EPO and invoice it has all the information necessary to approve or reject the purchase.

We use the term *certified delivery* to describe the mechanism of delivering encrypted information goods and

then charging against the customer's NetBill account, with decryption key registration both at the merchant's application and the NetBill server.

The NetBill transaction protocol also exhibits a number of other desirable features:

- *Support for flexible pricing.* By including the steps of offer and acceptance, we provide an opportunity for the merchant to calculate a customized quote for an individual customer. In the process we also generate signed messages that can later prove that there was a contract at the quoted price.
- *Scalability.* The bottleneck in the NetBill model is the NetBill server which supports many different merchants. Our transaction protocol minimizes the load on the NetBill server and distributes the burden over the many customer and merchant machines. Note that a single interaction with the NetBill server both verifies the availability of funds and records the transaction. It is not possible to have less than one interaction with the NetBill server [7].
- *Protection of user accounts* against unscrupulous merchants. In a conventional credit card transaction, the merchant learns the customer's credit card number and can submit fraudulent invoices in the customer's name. In a NetBill transaction, the customer digitally signs the EPO using a key that is never revealed to the merchant, thus eliminating this threat. Moreover, the customer has proof of the exact nature of the information goods received, providing evidence in case a dishonest merchant attempts to deliver faulty information goods.

NetBill account management

In this section, we discuss how customers and merchants can manage their NetBill accounts.

NetBill supports a many-to-many relationship between *customers* and *accounts*. A project account at a corporation can have many users authorized to charge against it. Conversely, an individual customer can maintain multiple personal accounts. Every account has a single user who is the account *owner*; and the account owner can grant various forms of access rights on the account to other users.

User account administration is provided through WWW forms. Using a standard WWW browser, an authorized user can view and change a NetBill account profile, authorize funds transfer into that account, or view a current statement of transactions on that account. Authentication and security are provided by treating account information as "billable" items. NetBill provides account information to users using the NetBill protocol. NetBill can be configured to provide this information for free or for a service charge, as desired.

Automating account establishment for both customers and merchants is important for limiting costs. (Account creation is one of the largest costs associated with traditional credit card and bank accounts.) To begin the process, a customer retrieves, perhaps by anonymous FTP, a digitally signed NetBill security module that will work

with the user's WWW browser. Once the customer checks the validity of the security module, she puts the module in place. She then fills out a WWW form, including appropriate credit card or bank account information to fund the account, and submits it for processing. The security module encrypts this information to protect it from being observed in transit. The NetBill server must verify that this credit card or banking account number is valid and that the user has the right to access it. There are a variety of techniques for this verification: for example, customers may telephone an automated attendant system and provide a PIN associated with the credit card or bank account to obtain a password.

NetBill costs and interaction with financial institutions

In a modern market economy, there are many forms of money, but two distinct poles typify the range of alternatives: *tokens* and *notational money*. Currency consists of unforgeable tokens that are widely accepted by both buyers and sellers as a store of value. In a cash transaction, the seller delivers goods to the customer while the customer delivers currency to the seller. Other projects are developing forms of electronic currency for network commerce based on unique digital bit strings.[8]

Demand deposit accounts at a bank are an example of notational money: on instruction (a check) by a customer, funds move from one ledger to another. A complex system involving intermediaries such as the Federal Reserve supports check clearing and settlements when the accounts are held at different banking institutions. Settlements can involve significant delays during which funds are not available to either party in a transaction. Notational accounts can have either a positive or negative balance, depending upon whether a bank is willing to extend credit to a buyer. For example, a credit card account runs a negative balance as the issuing bank executes instructions to transfer funds to a merchant's bank account.

Orders to transfer notational money are increasingly sent using electronic mechanisms: FedWire, automated clearinghouses (ACH), credit card authorization and settlement networks, and automated teller machine networks are all examples. NetBill also uses notational money. Because both customers and merchants maintain NetBill accounts, inter-institutional clearing costs are not incurred for every transaction. NetBill accounts provide a low cost mechanism to aggregate small value transactions before invoking a relatively high fixed cost conventional transaction mechanism. Customers move money into their NetBill account in large chunks (for example, \$50 - \$100) by charging a credit card or through an ACH transaction. Similarly, money moves from a merchant's NetBill account to the merchant's bank through an ACH deposit transaction.

NetBill accounts can be either pre-paid (debit model) or post-paid (credit model). In the prepaid model, funds would be transferred to NetBill in advance to cover future

purchases. If the user does not have sufficient funds to cover a particular transaction, that transaction would be declined. The amount of any prepayment is set by the customer, subject to minimums and maximums established by the NetBill operator. On pre-paid accounts, the system allows users to designate the balance at which she is prompted to transfer additional funds to Netbill. Because ACH transactions take several days to clear, a user prepaying her Netbill account through the ACH may not have immediate access to the funds. Funding through a credit card, while incurring larger transaction fees, allows immediate access to a prepayment.

In the credit model, transactions would be accumulated with payment to NetBill being triggered by either time (based on a pre-established billing period) or dollar amount (based on a pre-established limit). Because granting credit creates a risk of non-payment, higher transaction fees may be associated with credit, versus prepaid accounts.

The design space for electronic transaction systems has three crucial dimensions: risk, delay and cost. For immediate transactions, risks of fraud or non-payment can be dealt with in two ways: 1) incorporating an insurance fee proportional to the transaction amount, or 2) investing in sophisticated security systems with (high) fixed costs independent of transaction size. Credit card systems are of the first type, typically charging 1-3% of the value of the transaction, while FedWire takes the second approach. Delay can reduce risk by allowing verification of fund availability before committing a transaction, and by allowing batching to achieve economies of scale, particularly in interbank settlements. However, delay imposes opportunity costs when funds are not available until cleared.

NetBill is optimized for very low marginal transaction costs (on the order of 1¢) on small value transactions (on the order of 10¢.) Fixed networking costs are reduced by using the Internet with its substantial economies of scale, as opposed to a dedicated single function network. Because both customers and merchants maintain accounts at NetBill, most transfers are internal to NetBill; this reduces both risk and processing cost. When fund transfers outside NetBill are necessary, they can take advantage of aggregation, which spreads fixed transaction costs over larger sums. Use of ACH transfers and prepaid accounts minimizes risk at the cost of some delay before incoming funds are available; where NetBill offers deposits through credit cards, or grants credit itself, the risk increases and must be passed on to customers as higher fees.

NetBill keeps other costs of operation low by: automating all account administration functions; using techniques like certified delivery to reduce the incidence of complaints and customer service costs; and using a modern distributed processing approach for the core NetBill processing system.

An example of NetBill with Mosaic

Because WWW browsers and servers are a *de facto* standard for distributing information over the Internet, we have created a prototype implementation of NetBill that allows for billing of WWW transactions. Rather than link the NetBill libraries with a WWW browser and http server respectively, we have enabled commerce with no modification to either the browser or the server. Our design introduces two entities in order to support the exchange of money for goods: the Money Tool and the Product Server. The Money Tool runs on the customer's machine and works with a Mosaic browser. It allows the customer to authenticate, select accounts, approve/deny transactions, and monitor expenditures. The Product Server, which incorporates the till libraries, works with the http server to sell information products.

When a user clicks on a product in a product server's catalog, the server returns a special file with a mime type containing information about the server's identity, the product to be ordered, and the port number of the product server. This mime type spawns a "helper" program in the same way that jpeg, sound, and mpeg files currently do. The spawned program communicates the contents of the file between Mosaic and the Money Tool.

The Money Tool acts as the customer's application in the NetBill transaction protocol described above. After it receives and decrypts the goods, it uses the remote control function of Mosaic to cause the browser to display the received information. Besides implementing the steps in the protocol, the Money Tool provides a number of useful functions to help the user manage transactions (Figure 3):

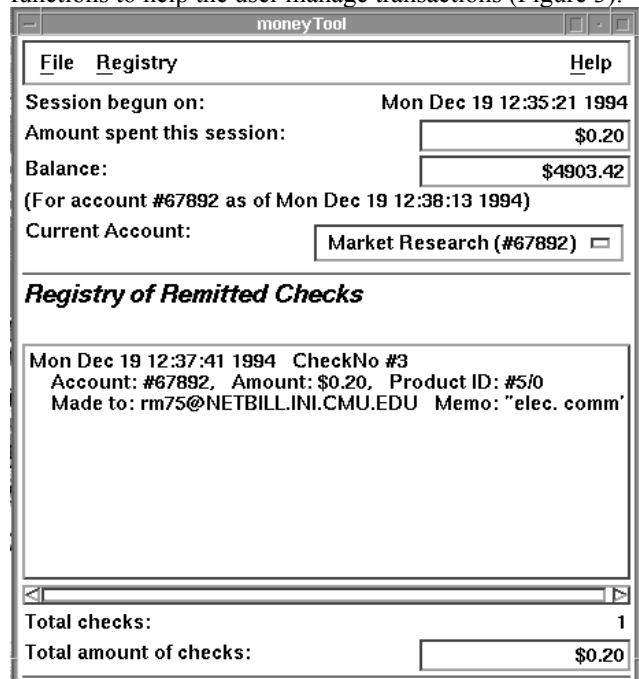


Figure 3: The Money Tool

- it provides an authentication dialog window

- it provides a running total of expenditures in the current session and the current balance in the user's NetBill account
- it provides a listing of all EPOs processed in the current session
- it can be configured to automatically approve expenditures below a threshold
 - it can be used to retrieve the product encryption key from NetBill in the event of failure of a merchant host.

Spyglass has recently proposed a standard API for Security Plug-in Modules for WWW browsers [9]. In the future we expect to integrate the Money Tool with the browser using this mechanism.

In the current implementation, the initial request for goods to the http server causes the server to run a script that writes information about the request to a temporary file at the server. When the Product Server receives a request for a price quote from the Money Tool, it must access the server's database to determine the price quote based on the customer identity. If the quotation is approved, the product server finds the goods using the information saved by the http server and completes the NetBill transaction protocol,

Additional issues

As described above, NetBill is well suited for supporting commerce in information goods. However, the NetBill model can also be extended in a variety of ways to support other types of purchases. For example, NetBill could be used equally well for conventional bill paying. A customer could view a bill presented as a Web page; instead of buying information goods, we can think of the customer as buying a receipt for having paid the bill.

If the product to be bought is a one hour movie, it is likely that the customer will want to stream the data directly to a viewer, which conflicts with NetBill's model of certified delivery. We are exploring alternative approaches such as using the standard NetBill protocol to periodically buy a key for the next N minutes of an encrypted video stream.

We are also exploring the software rental application. A software vendor could incorporate the checkbook library in any arbitrary application software. Periodically, the software would ask the user to approve the purchase of a key for the next month's operation. (This requires

mechanisms to prevent the software vendor from including a Trojan Horse designed to capture a renter's password.)

Acknowledgments

Much of the development of NetBill has been done by students in project courses taken as part of Carnegie Mellon's graduate program in Information Networking. We thank all of those students for their help and ideas. Support for our research was provided in part by a grant from the National Science Foundation.

Notes

For more information on NetBill, including a fuller version of this paper, please look at our WWW page at <http://www.ini.cmu.edu:80/netbill>.

1. Porat, M., *The Information Economy* (US. Office of Telecommunications, 1977)
2. *New York Times*, June 7, 1992
3. Separately, we are researching means of embedding a unique watermark in each copy sold which would allow illegal copies to be traced to the source.
4. In the special case of free information, we can optimize our protocol still further.
5. In separate work, we are designing pricing servers that can handle a very broad range of pricing strategies.
6. Rivest, *The MD5 Message Digest Algorithm*, April 1992.
7. In theory, one might bundle several transactions together and have them all processed as part of one interaction with NetBill. However usage data collected from Carnegie Mellon's Library Information System indicates that in the majority of cases, users contacting the library are looking for a single item, suggesting that bundling would not be appropriate. Cf. O'Toole, K., *The Internet Billing Server: Transaction Protocol Alternatives*, Carnegie Mellon Information Networking Institute Technical Report TR 1994-1.
8. Chaum, D., "Achieving electronic privacy", *Scientific American*, **267**, No. 2, pp. 76-81, 1992
9. Jeff Hostetler, "A Framework for Security," 2nd WWW Conference, Chicago, Illinois, October, 1994.