QuickPay: Protocol for Online Payment

Graduate Project for CS 298

Submitted by Jian Dai
Advised by Dr. Mark Stamp

Reviewed By: Dr. Chris Pollett
Reviewed By: Dr. David Blockus

May 9, 2006

ABSTRACT

Even before the Internet bubble, online payment was viewed as a promising technology [4, 22]. But in spite of the potentially large demand, such systems have failed to become a major method of online financial transactions. Based on the analysis of existing online payment systems, we propose QuickPay as a new online payment protocol.

QuickPay is a middleware framework built for online payment and has the potential to be used for other online financial transactions. The purpose of QuickPay is to serve as a highly effective, user-anonymous, secure infrastructure for online transfer of financial data at a low transaction cost. In this report, we will introduce the underlying motivation and provide a detailed design of the QuickPay system. We will describe in detail the QuickPay business and technical goals, general architecture, major players, and relationship among the players. To show how QuickPay works, we implement two general online payment protocols (micropayment and Regular Payment) and other basic administration and communication protocols. After evaluate QuickPay's security and performance, we believe that QuickPay can provide a more effective and more secure framework than other existing online payment systems.

1. INTRODUCTION

The Internet offers an unprecedented opportunity for access to digital content and other new online business activities. During the Internet bubble era (about 1995 through 2000), most digital content on the Internet was free, with merchants expecting to eventually cover their costs through advertisements. This business model is now widely discredited, and it is generally agreed that online merchants need to charge for their products and services [14]. Consequently, electronic financial transactions must be accommodated.

Online payment systems are designed to facilitate online purchases involving variety of amounts of money and micropayment is one of the most important features for these systems. Some situations where micropayment naturally arises include downloading a news article, a recipe, a song, requesting information from a specialized database, or even buying a lottery ticket [5, 10, 18, 19].

Unlike traditional payment, the requirements of micropayment have at their root challenges, which are faced by businesses over online financial transactions. These challenges may include [11, 15, 22]:

- Trust and Anonymity. Over the Internet, there is a physical separation between customer and vendor and this fact makes it difficult to establish trust between the two. Meanwhile, the customers using online payment usually have only a transient connection with vendors and are not willing to tell the vendor too much about himself/herself.
- Instant Payment. Typically, online payment, specifically micropayment, is used for digital content download or online services. For example, a customer may use it to

pay for an article, download an audio record, or play an online game. Unlike the traditional orders, which usually don't need to be delivered immediately and where the vender has enough time to verify payment before delivery, these contents or services must be delivered to the customer instantly and the deliveries cannot be revoked. After the customer downloads article, there is no way for the vendor to take it back)

- Low Processing Cost. Since online payment, specifically micropayment, tends to deal with small amount payments, the processing cost becomes of special importance. As a matter of fact, it is the cost concern that makes micropayment different from other payment processes and many micropayment systems have been developed just to answer this challenge.

At present, there are many online payment systems have been developed. Based on their design principles and main characters, we can roughly classify them into three categories.

1.1. Token-based Systems

Such systems borrow the concept from physical currency and try to create "virtual" coins, which can be used by everybody [3, 4, 8, 9, 17, 23, 26, 27]. Generally, a token-based system has following step in its payment cycle:

1. A high-credibility broker creates a so-called "digital token" based on a special cryptography algorithm (a special hash function in most cases).
2. The digital tokens are sold to customers who want to use them for online shopping.
3. When a customer buys something online, he/she pays vendor with those tokens.

4. Once the tokens are validated, the vendor will deliver the content and collect the tokens.
5. At the end, broker will reimburse the vendor for the tokens they collected.

Besides, the token-based systems also face some common business and technical challenges [3, 13]:

- *How do the customers get these tokens?* Based on this problem, micropayment systems can be divided into prepaid systems or postpaid systems. In a prepaid system, the customers need to purchase the token first. In a postpaid system, the customers "borrow" the token on a credit basis and will pay later after they spend their tokens.
- *How to create tokens?* This question is the core part for such systems. The created tokens must be very easily to be verified and very difficult to be faked.
- *How to verify tokens?* Once a vendor receives a token, it needs to check that token's validity. A good online payment system should provide a very efficient validation function for vendors. At present, hash function is the most popular way to provide this kind of feature.
- *Does the broker provide online support?* Online support can make vendor's job easier with regard to token validation and increase the overall security of the system. But this does not come without a cost. Online support could be very expensive to the broker. Meanwhile, the broker could be a bottleneck and single fail point for the whole system.

With careful design and creative token creation and verification algorithm, most of proposed token-based systems can satisfy above basic business and technical requirements. On the other hand, in our opinion, these systems still have some inherent business and technical weaknesses, which make them less likely to be widely accepted by the Internet community.

Cost miscalculation. As we mentioned, the cost is the primary concern for micropayment. Token-based systems are carefully designed to reduce the marginal cost for payment transaction and they generally did a good job here. But there are some other kinds of costs, which are ignored or miscalculated by these systems. For example, *Overhead cost could be miscalculated*. In token-based systems, some one-time events may occur and the cost of these events is usually high. If these one-time events can lead to enough regular transactions, this cost will be shared by those transactions and won't be a big problem. And this is this situation that is assumed by token-based systems. But this assumption may not always be true. For example, set up commitment for PayWord chain is an expensive process (because of public key operation involved here) and the system assumes a chain can be used for many payments. But a more likely case of micropayment is that a customer surfs a lot of websites and intends to have a little purchase on single web site once a time. In this case, the chain of PayWord will be only used by a few payment transactions and its cost cannot be justified.

No support for regular payment. A major debate over the future of micropayment is whether a special online payment system dealing only with small size payment is really needed. No matter what conclusion this argument finally reaches, it is always ideal if an online payment system can support both micropayment and regular payment without big problems. Unfortunately, token-making systems generally cannot satisfy this demand. The general problem is that the size of a token usually is small and fixed so that it is difficult to use for regular payment. For example, if we make token size of 5 cents, we would need 100 tokens to pay for a $5.00 purchase. It could be highly inconvenient and potentially expensive to pay for an order with even higher value with token.

Compatibility absence. The Internet is a decentralized system. It is very difficult for a single micropayment to dominate the whole system. Therefore, a good online payment system should be an open system, which can work with other systems. Unfortunately, none of existing token-based system can satisfy such a demand because their unique token-making algorithm can not work with each other.

1.2. In Account-Based Systems

In such systems [1, 2, 12, 21], customers and vendor both need to register accounts on broker's site first. After login, the customer can only purchase the contents or services from the vendors, which are the members of that broker. When the customer order from a vendor, the amount of payment will deduct from his/her account and added to that vendor account. In account-based system, the actual money transaction does not occur with every individual purchase until a member (as customer or vendor) deposit or withdraw money from his/her account.

Generally, account-based systems have some shared characteristics:

- There is a specific company who hosts the whole system. Anybody who wants to use that payment system must register an account and become a member (or partner) first.

As a result, all the members (partners) consist of a closed community. The payment transaction can only occur within the community.

- Unlike a token-based system, in which the roles of vendor and customer are generally distinguished, an account-based system, in theory, treats all its members equally. A member can be a vendor (when he/she sells) as well as a customer (when he/she buys).
- Since all the members have accounts in the host's system and payment transactions occur between members, the transactions can be done inside the host's system. As a result, the cost for transaction security is very low.
- In account-based systems, the member authentication is the central security issue. Meanwhile, the host must provide online support and its site becomes the bottleneck for the whole system.

Generally, an account-based online payment system enjoys the following advantages:

No specific hardware/software requirement. Usually, an account-based system does not have specific software or hardware requirement for a user to join in. a user can log in and accesses the system.

Anonymity between members. In an account-based system, the host plays mediator between members (as seller and buyer). During a payment process, the host will handle the transaction and the buyer and seller don't know and don't need to know to each other.

Very low transaction cost. This advantage has been mentioned earlier as one of common characteristics shared by account-based system.

Easy to handle forgery situation. In account-based systems, since system host handle the whole transaction and has a good knowledge about customer and vendor (by holding their accounts), forgery payment can be easily detected and prevented.

Besides all the advantages mentioned above, account-based systems also suffer some drawbacks. First, like other membership-based applications, account-based online payment systems use username/password to authenticate a member, a well-known weak security measure. Secondly, account-based systems are generally hosted and managed by a specific company and the transaction will occur inside that company's system. Therefore, it is impractical to have a transaction across systems. Meanwhile, for business reasons, big E-commerce companies would rather have their own system than use another company's system. This concern makes account-based system difficult to expand its customer and partner basis. (Even a system that is as successful as PayPal still mainly depends on its partnership with eBay.) Besides these, the host's server in an account-based system is regarded as a bottleneck and single fail point for the whole system. Every transaction must go through host's server and all the users' information is saved there. If a hacker can break the host's server, he can crack the whole system. Meanwhile, the load of host's server could become a bottleneck for the whole system and make the system more difficult to be scalable.

1.3. Protocol-Based System

Usually, a hardware/software vendor proposes and endorses such systems with a set of predefined protocols (may be associated with specific hardware and/or software) [6, 20, 25]. Anybody who adopts this protocol should be able to communicate with and establish a trustworthy relationship with corresponding parts. As a result, the micropayment transaction can be delivered between the correspondent parts as long as they both use the same protocol. At times, a protocol may be associated with a specific hardware and/or software.

A Protocol-Based Payment system is a different implementation of an online payment service. Unlike the payment system we introduced above, whose design goal is to provide a specific means of online payment, the design goal for protocol-based payment systems is to provide a general infrastructure to handle the most basic task of online payment and enable different payment systems that can work together by using specific protocols. Protocol-based payment systems are still new and currently have not been commercially used yet.

Compare to other system protocol-based systems has some distinguish advantages. Designed as open system. Protocol-base systems provide standard infrastructure and protocols, which can be implemented by any interested party. Therefore, such systems can accommodate different payment types. In theory, they can easily build a big user basis because vendors and customers with one payment system will be able to work with others who may be using different payment systems via a standard infrastructure. Besides, Protocol-based systems generally only handle very basic functions for online payment transactions and provide standard APIs for these functions. Different payment service providers can implement these APIs in optimal ways based on their own situations. Meanwhile, since Protocol-based systems already include functionalities for the basic online payment functions, payment service providers do not need to rebuild such functions again by themselves. This should be able to save them a lot of system developing time and money.

On the other hand, protocol-based systems also have their own problems. Usually, protocol-based systems require a big investment of money and time in order to set up the system in the first place. The pre-operation overhead investment is big even though the operating cost per unit for such systems can be very low. It is kind of like a telephone company in the old times. The setup cost for a telephone network is very big but afterwards, the margin cost for each communication is very low. The other problem with such a system is its technological difficulty. Since the goal of such a system is to provide a common infrastructure for a variety of payment types, it must be able to deal with widely different cases in a common way. And it also needs to have enough flexibility to accommodate emerging new payment types. As we can understand, from a technological standpoint, it is not an easy job at all.

Since the mid-1990s when the Internet began to take off, a lot of different online payment systems have been introduced, but there are few, such as PayPal, successes. So far, traditional payment methods (credit cards, membership fees, and so on) are still the dominant payments in E-commerce. Besides the technical challenges (e.g. authentication, date integrity, low-transaction cost, data persistence, and so on), one of the major obstacles faced by an online payment implementation is customer acceptance [16]. Because of the high overhead cost and the low transaction fees, an online system can achieve business success only if there are enough customers to use it. Unfortunately, most of existing online payment applications have limits on customer acceptance due to their design defiant on security and performance concerns. Therefore, we believe that there is requirement for a different online payment system and QuickPay is designed for this requirement in this report.

In this report, we propose QuickPay as a generic online payment application with optimal tradeoff among security, effectiveness and costs, and one can be widely used for a variety of commercial activities.

Based on our studies for existing online payment systems, we first define the business and technical requirements for QuickPay in section **Business And Technical Background**. We believe these requirements are the most critical and important and QuickPay won't be successful until it can satisfy all and each of these requirements.

After then, the overall architecture of QuickPay has been described in section **Architecture Overview**. In this section, we will try to give a whole picture of QuickPay's high-level infrastructure. We are also going to explain the basic principles of QuickPay protocols, message format, major player roles and the relationship between them, and other architecture characteristics and terminologies.

In section **Protocol Description**, we will provide detail description for individual protocols defined in QuickPay. In this section, we won't cover all of available protocols but only the most important ones. Using these protocols as examples, we explain how QuickPay protocols are designed and how they work in particular business scenarios.

In section **Security and Performance Analysis**, we will evaluate QuickPay's security and performance concerns in both architecture level and protocol level. We are also going to describe how QuickPay answer the challenges faced by online payment systems and satisfy its own business and technical requirements. At the end, we will summarize the advantages of QuickPay over existing online payment system.

2. BUSINESS AND TECHNICAL BACKGROUND

2.1. Business Requirements

Based on our best understanding about general processes of online payments and the study of how current existing and proposed online payment systems work, we believe following business requirements are the most critical to QuickPay's success.

Customer anonymity [24]. From the vendor's point of view, it means that the vendor does not have to keep the customer's information and to identify the customer by itself during the online payment process. As a result, the vendor will be able to serve for a wider base of potential customers. From the customer's point of view, anonymity makes his/her online shopping more convenient (he/she does not have to log in everywhere), more comfortable (e.g. a customer may want to buy an adult video and not want to leave any record), and less worrisome for security (his/her personal information is not exposed to and potentially abused by a malicious vendor).

Support for small-size payment (micropayment). As we mentioned above, micropayment can be used for activities such as downloading music, articles or video, getting expert advice on a special issue, and so on. Since such activities are the very important online business and there is few existing online payment system can really support it, QuickPay should be able to support micropayment in order to fill this market niche

Forged payment prevention. As we know, forged payment prevention is the most basic requirement of any payment systems. Like other online payment systems, QuickPay should be able to addresses following concerns.

- *Transaction Validation*. For each online payment transaction, QuickPay should ensure the payment transaction is from the right customer to the right vendor and the data won't be corrupted or manipulated during the online transaction process.
- *Prevention of double spending*. QuickPay should prevent customers from using single payment commitments to purchase from multiple vendors and will also prevent vendors from double charging customers for a single commitment.
- *Guarantee of final payment*. In QuickPay, if a vendor gets a commitment from a customer and that commitment is validated by the system, the vendor is guaranteed to get final payment for that commitment.

Support instant order delivery. This support is regarded as a big advantage of token-based online payment systems because digital content, which is bought with micropayment, usually needs to be delivered immediately [7]. Since supporting micropayment is one of the most important components, QuickPay also supports combined order and delivery.

2.2. Business Assumptions

Besides to cover all of business environment, with which typical online payment systems deal, we make QuickPay focuses on the most important and most common scenarios of online payment processes. By doing this, we expect to simplify the complex of overall system and accelerate the development process and make QuickPay easier to be accepted by Internet community. As a result, we make following assumptions about the business environment, under which QuickPay works

Most online payment occurs on customer's private domain. Most likely, in our opinion, customers do online shopping from their dedicated domains, such as home computers, office computers assigned to them, their laptops, or wireless phones instead from public domain, such as public libraries, school labs, or Internet bars. Therefore, bonding customer identification with specific machine domains can be considered as an accepted convenience for most of online payment processes.

Broker has much less motivation to cheat over a transaction than customer. This assumption is easier to understand if we look at the New York Stock Exchange. Over there, only registered brokers can trade stock on behalf of their clients. Since these brokers do not have any gain/loss on each transaction but will get high commissions for their work, they almost never make forged trades. As a result, those brokers can trade stocks in an extremely informal but highly effective way during open market time and then get all the deals settled after the market closes. For the same reason, if we can encourage some brokers with high reputation to pay for their clients on online payment transactions, we can expect these brokers won't cheat on purpose. As a result, much security cost to prevent customer's cheating can be saved.

2.3. Technical Requirements

Based on our observation and analysis of exiting online payment applications, we believe that QuickPay, as an optimal online payment application, should satisfy the following technical requirements:

<u>Openness.</u> With the development of Internet society so far, we understand that many Internet users may already have been working with some online payment systems. Besides work for its own users, QuickPay is required to accommodate users with other online payment systems.

<u>High scalability</u> QuickPay is required to be a highly scalable system. This goal is achieved with following requirements:

- Unlike most traditional token-based and protocol-based payment applications, whose acceptability has been constrained by highly special technical requirements, QuickPay is required to only use standard, platform- and language-independent technologies.
- Unlike account-based applications, which require complicated and costly users' registration and authentication processes, QuickPay requires a "no-knowledge" relationship among users. Therefore, the user can join and leave the system independently.

- QuickPay is required to bundle all of associated functions for a special payment process together as a protocol and protocols are independent of each other. Therefore, a user can choose to use only a subset of existing protocols based on its own business consideration.

High security and low transaction cost. Transaction security is the core requirement for any online payment application. QuickPay is required to have a higher (or at least equal) security standard on all of the involved security issues. Besides achieving a higher standard security, QuickPay is also required to achieve its security goals at an equal or lower transaction cost than existing online payment systems.

Standard User Interface QuickPay is required to have a standard user interface for all the End Users, no matter which broker these End Users work with. We believe this requirement is very critical for QuickPay because it can dramatically improve user's experience with QuickPay and encourage them to use QuickPay for long time.

3. ARCHITECTURE OVERVIEW

3.1. General Architecture Description

QuickPay is designed as a middleware application based on HTTP, Web Service and other related standard protocols. As shown in Figure 3.1.1, QuickPay's architecture has the following characteristics:

Openness and Plug-in Capability In QuickPay, the major players (player roles have been described in section 3.2) can freely and independently join system as long as they implements its role in QuickPay's protocols. For example, a Vendor who wants to only use QuickPay's micropayment protocol can join the system as soon as it implements the functionalities required for a Vendor in that protocol.

Indirect trusted communication One of the core ideas of QuickPay's design is that the communication between trusted parts is much more reliable. In QuickPay, players rely on their trusted partners to provide them with correct relevant data during an online payment transaction. For the players (such as Vendor and customer), between which there exists no direct trusted relationship, QuickPay establishes a trust chain to handle the communication between them. We called this mechanism as an indirect trusted communication.

Loose relationship between players In QuickPay, a player has not a permanent bond with any other player. In other words, there are many-to-many relationships between different types of players and players can easily add/drop their relationship with another player over time.

Protocol independence. As mentioned, QuickPay's application protocols are designed as independent from each other. As a result, a player can support only a subset of all available protocols based on its own business requirements. On the other hand, when QuickPay introduces a new protocol in the future to explore new business opportunities, the existing system won't be affected and new protocols do not have to involve the compatibility problem with legacy protocols.

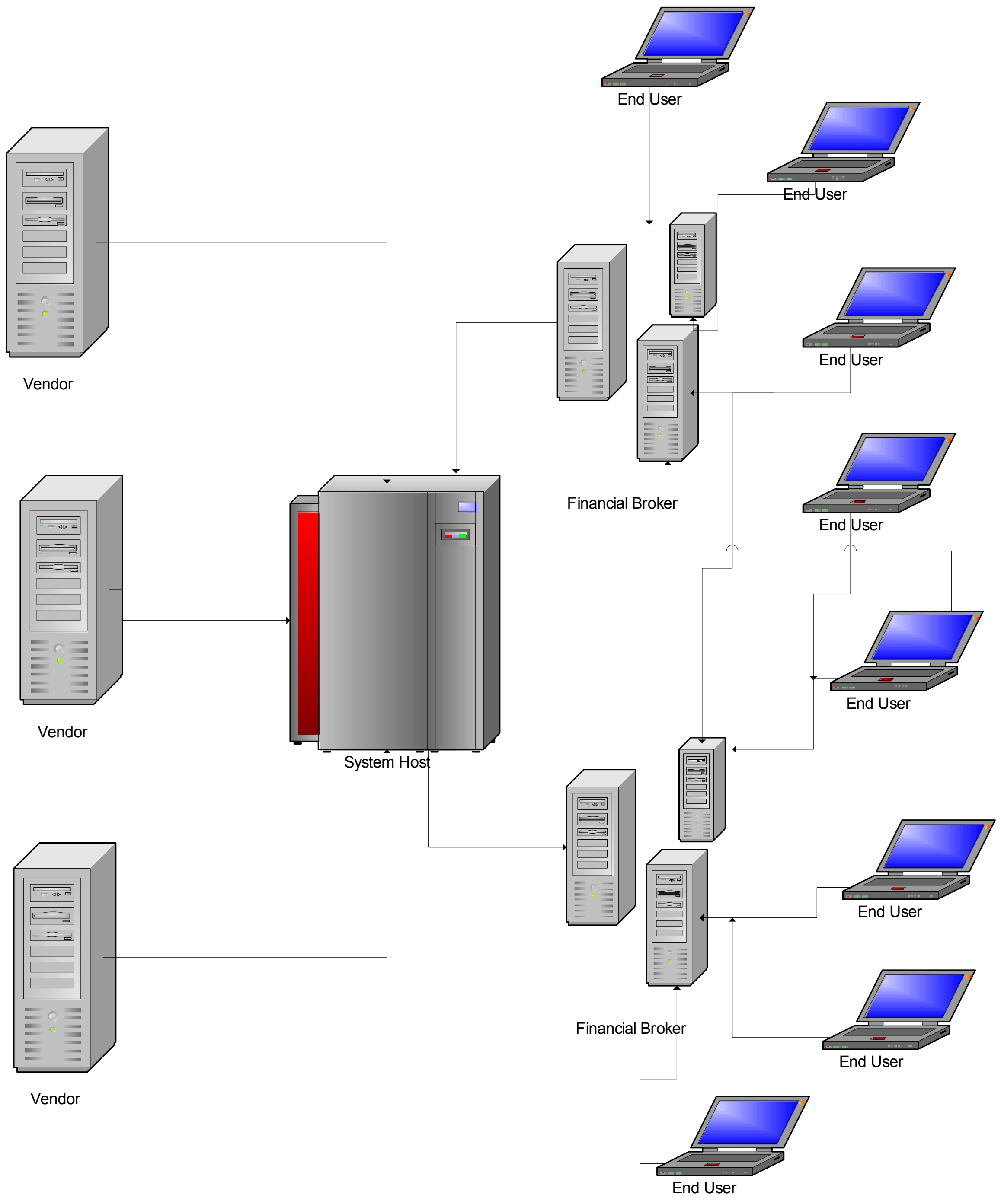


Figure 3.1.1

3.2. Player Roles

QuickPay defines four types of players for the whole system. From a business point of view, each type of player may correspond to a special kind of physical participant involved in an online payment transaction. From a technical point of view, each player represents a logical role defined by a set of functionalities in QuickPay's protocols. In theory, a physical entity (such as a website) may play multiple roles as long as it implements all the functionalities required for those roles.

System Host. The System Host is the central controller and administrator for the whole system. System Hosts perform the following functions:

- **Mediate the information exchange between other players**. In QuickPay, the System Host is the only player who can be trusted by everybody and has QuickPay-related authentication information for everybody. In general, a player does not have to keep any other player's authentication information and should always rely on System Host to provide such information. Meanwhile, the System Host also plays as message exchange center for other players (see detail discussion in section 4.3).
- **Monitor the payment process and play as a fair third part judge**. Unlike the existing online payment applications in which the payment transaction occurs only between payer and payee, QuickPay requires the System Host to "watch" payment transactions. In other words, QuickPay's payment transaction involves three parts, and the System Host should keep track of each payment procedure (see detail discussion in section 4.2). As a result, when an argument occurs over a payment, the

System Host, with the information it gets from monitoring the payment process, can play as a fair judge to settles it.

Vendor. The Vendor is anybody (usually an E-commerce website) that is selling products (tangible or intangible) online and intending to use QuickPay to collect payment. To join QuickPay, a Vendor only needs to get a unique Vendor identification number from the System Host and support all the required functionalities for the Vendor in QuickPay's protocol. Theoretically, the System Host knows very little about Vendors in QuickPay even thought it might need Vendor to register account.

Financial Broker. The Financial Broker is a financial institution or brokerage that is willing to pay for an online purchase on behalf of its customers (as End User in QuickPay). Unlike Vendor, Financial Broker is a trusted partner and is expected to have a long-term very close relationship with The System Host. In some ways, QuickPay works as a club for its Financial Broker members, just as the New York Stock Exchange works for its seating Brokers. QuickPay, via its System Host, helps its Financial Broker members to provide their own customers with online payment service and get commissions in return. The System Host then shares the commission by charging a service fee from its Financial Broker member. In QuickPay, each End User must be a customer of one or several Financial Broker(s). QuickPay expects that the Financial Brokers maintain all the information about End Users (as their customers) and know how to communicate with them securely.

End User. The End User is generally regarded as a person who wants to use QuickPay to pay his/her online purchase. From a business point of view, End User is a customer of QuickPay's Financial Broker as well as an individual user of the QuickPay system. From a technical point of view, End User is a Wallet bonded with a machine domain (desktop, laptop, Palm, Internet Phone, or other), which is endorsed with the QuickPay system. In this way, End User can be thought of having three parts: web browser (I.E., Netscape, Palm, or Internet Phone), Wallet, and Paycard. Among them:

Wallet. Wallet is a browse plug-in software component provided by the System Host. It can 1) provides standard graphic user interface on End User's client machine for all the QuickPay functionalities; 2) maintains necessary client-side data; and 3) handles all the online communication between End User and other parts.

Paycard is a Wallet plug-in software component provided by the Financial Broker. A Wallet can have multiple Paycards for different Financial Brokers and should have only one active at any time. Paycard is used to provide APIs for Wallet to communicate with its host Financial Broker.

3.3. Relationship among QuickPay Players

As mentioned above, QuickPay has four major types of players. The relationships between these players are shown in Figure 3.3.1 and briefly described here:

The System Host and Financial Broker has the closest relationship in the QuickPay system. Instead of working for End Users directly, QuickPay is designed to work for a Financial Broker, which in turn, works with the End User directly. As result, the number of Financial Brokers supported by QuickPay is limited. A Financial Broker should be a decent business entity, such as banks, credit card issuers, loan agencies and other financial institutions. It should have a long-term relationship, which is legally bonded with formal contracts. As a result, the System Host has fully trusty on Financial Brokers.

Relationship between System Host and End User is based on Wallet. Wallet is a client side software component issued by the System Host. Each Wallet should have a unique identification assigned by the System Host. From the user's point of view, Wallet is the central point of the QuickPay system in which users can communicate with other players. In order to be able to use QuickPay, an End User must install Wallet and register it on the System Host first. This registration is about the Wallet but the user himself/herself. In other words, after registration, the System Host will have all the information about that particular Wallet but about that user himself/herself. This design has two purposes. First, it means that the System Host does not know who is going to use that Wallet but only knows from which machine the user can access the QuickPay system via that Wallet. Secondly, from the System Host's point of view, End User means an active Wallet that has been installed and registered instead of real people. In the real world, the same user can be regard as different End Users if he/she has several registered Wallets while several customers may be regarded as a single End User if they share the same Wallet.

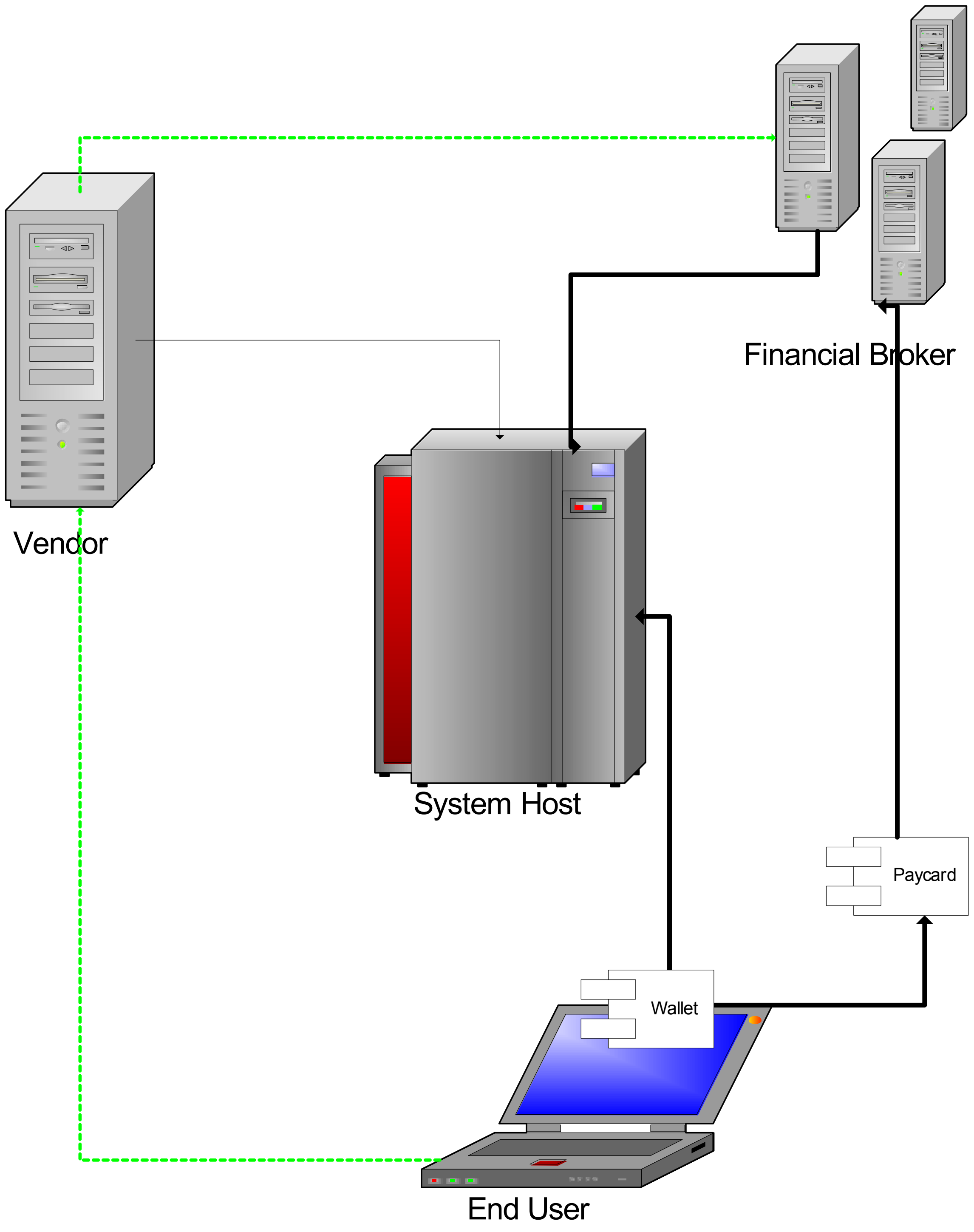

Financial Broker
Vendor
System Host
Paycard
Wallet
End User

Figure 3.3.1

Relationship between Financial Broker and End User is based on Paycard. In QuickPay, Paycard is a client-side component developed by the Financial Broker. Paycard plugs in to Wallet and is used as a bridge between End User (Wallet) and Financial Broker. QuickPay defines the standard application interface for the communication between Paycard and Wallet. Each Financial Broker can have different implementations with these APIs and can have its own way of handling the communication between Paycard and the Financial Broker's server. One Wallet can have multiple Paycards, which come from different Financial Brokers and should have only one of them set as an active Paycard, which is actually used in the payment process. The User can use Wallet's configuration function to pick one of the available Paycards as the active Paycard.

The vendor has a loose relationship with System Host. Unlike the Financial Broker, The vendor is highly scalable and QuickPay can support an unlimited number of Vendors. In theory, any one who wants to sell its product or service on line can join QuickPay as Vendor and use QuickPay to collect payment. In practice, to become a QuickPay Vendor, an online seller only needs to meet two simple requirements: register on the System Host to get a unique Vendor ID and support Vendor functionalities in QuickPay's protocols. The Vendor registration process is pretty simple. The Vendor only needs to provide the IP address and port number of its server, which is going to be used for QuickPay communication.

End User and Financial Broker have order-based one-time relationship with Vendor. End User does not have any relationship with a Vendor until he/she wants to pay for that

Vendor with QuickPay. In QuickPay, the relationship between End User and Vendor sustains only during the order process. Unlike credit card payment, the Vendor cannot collect and keep any personal and financial information about the End User except for the order payment itself here. With this design, Vendors won't need to maintain End User's information so that it theoretically can work with any End User. Meanwhile, End User won’t worry that the Vendor could abuse his/her personal or financial information after a one-time shopping.

4. PROTOCOL DESCRIPTION

As we mentioned in the architecture overview, QuickPay is composed of a set of independent protocols. In this section, we will introduce protocols currently supported by QuickPay in this project.

4.1. Administration Protocols

The purpose of Administration protocols is to help related players to set up or update configuration information so that they can run QuickPay application protocols effectively and securely. Generally, Processes supported by administration Protocols run on only when some special events occur, such as installation, removal, updated information, etc. Since administration protocols run much less frequently but are more critical for overall system security than application protocols, the implementation of these protocols should have emphasis on security instead of performance. In QuickPay, support of Administration protocols is required for System Host, Financial Broker, and End User but for Vendor. With this design, QuickPay becomes highly scalable to Vendors.

In our current design, a set of administration protocols has been defined (see appendix B). "Add Paycard" is one of the most important administration protocols and serves for several purposes. First of all, in order to prevent from forged Paycard issued by hacker, adding Paycard component must be certified by its host Financial Broker and its authentication should be verified with this protocol. As we mentioned earlier, Paycard represents an account in its host Financial Broker, and that Financial Broker will pay any payment request sent from that Paycard on behalf of the owner of that associated account. This protocol can be used to associate Wallet with a particular account in Financial Broker. At the same time, the Wallet's information is also provided to Financial Broker with this protocol.

This protocol is composed of three messages (See table 4.1.1):

Seq. No	**Msg. Name**	**Sender**	**Receiver**	**Purpose**
1	Confirm Paycard	Wallet	Broker	Verify adding Paycard's authentication
2	Take Question	Wallet	Broker	Take questions regarding user's personal/financial data from Broker.
3	Answer Response	Wallet	Broker	Return user's answers back to Broker

Table 4.1.1

4.2. Application Protocols

Application protocols are the workhorse of QuickPay system. One or several bonded application protocols are used to handle a special online payment or financial data transaction. The protocols supporting the same scenario are highly related and should work together. On the other hand, protocols for different scenarios are totally independent to each other. Therefore, A player can selectively support a subset of application Protocols based upon its own business situation. Since application Protocols run on a regular basis and they are supposed to bring heavy traffic to a system. Therefore, the performance is the most important concern for those Protocols. QuickPay shifts some security burdens to administration Protocols in order to improve the their performances.

In this project, we only define application protocols for micropayment and Regular payment but more protocols should be able to add later independently.

4.2.1. Micropayment Protocol

In order to handle micropayment process, this Micropayment protocol must runs at very low transaction cost. As mentioned in section of introduction, the micropayment is most likely to be used to pay for business that requires the support for instant order delivery. This protocol is designed to meet this business requirement. Besides, this protocol is also design to allow the System Host to monitor the transaction process and support QuickPay's parallel payment pattern (will discussed in Section 4), which is used to achieve high security at a low cost.

The communication and major messages for this protocol are shown in Table 4.2.1.1., and Figure 4.2.1.1

Seq. No	Msg. Name	Sender	Receiver	Purpose
1	Get Order	Wallet	Vendor	Get micropayment order for Vendor
2	Confirm Order	Wallet	Wallet	Get user’s final confirmation to process order
3	Inquiry Payment	Wallet	Broker	Ask Broker to pay for the order
4	Request Order Delivery	Wallet	Vendor	Send Broker endorsed order back to Vendor
5	Check Commit	Vendor	Broker	Check Broker’s commitment
6	Register Commit	Broker	Host	Register Broker’s commitment
7	Register Delivery	Vendor	Host	Register Vendor’s delivery
8	Register Order	Wallet	Host	Register End User’s reception of order

Table 4.2.1.1 Messages for Micropayment Protocol

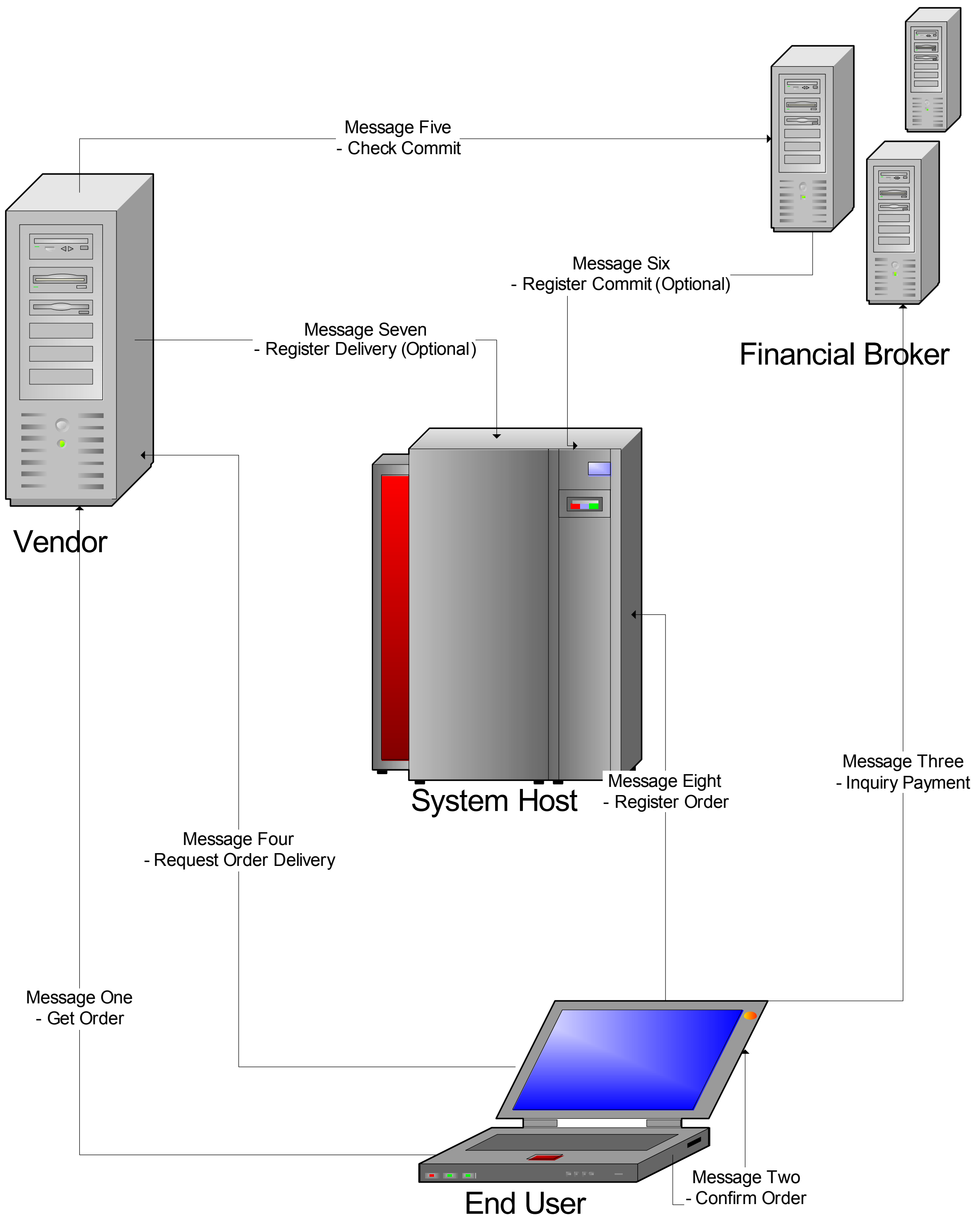


Figure 4.2.1.1. Message Flow for Micropayment Protocol

As we can see, Micropayment protocol is designed with emphasis on performance rather than security. In this design, 1) all of the online transactions are public and plaintext format, and there is no security implementation over them; 2) The payment commitments of the Financial Broker are temporal, which will be finally settled in Batch Settlement Protocol; and 3) System Host does not get involved in the payment transaction directly but does monitor the whole process indirectly.

4.2.2. Batch Settlement Protocol

This protocol is used to work with Micropayment process to handle the whole Micropayment scenario by settling all pending micropayment orders processed in micropayment protocol. With this protocol, Vendors will first try to get formal signed payment commitment from the Financial Broker. If Vendors cannot get Financial Broker's signed commitment for some orders, they can then turn those orders to System Host for approval. In QuickPay, an order will be validate if it can be either committed by Financial Broker or approved by System Host. Meanwhile, unlike micropayment protocol, this protocol is called for once a while and works as final settlement. Therefore, security is the major concern here.

The major messages for this protocol are shown in Table 4.2.2.1.

Seq. No	Msg. Name	Sender	Receiver	Purpose
1	Settle With Broker	Vendor	Broker	Settle micropayment orders with Broker
2	Settle With Host	Vendor	Host	Settle micropayment orders with System Host

Table 4.2.3.1 Messages for Batch Process Protocol

4.2.3. Regular Payment Protocol

In order to handle a regular payment (with high order value), Regular payment protocol is designed to process payment transactions with higher security at a reasonable cost. In this protocol, security becomes more important than transaction cost. With our design, regular payment is most likely to be used to pay for some tangible products (such as books, clothes, toys, and so on), and the purchase cannot be delivered immediately to End User online. Therefore, this protocol is not supposed to support instant order delivery and the underlying order can be revoked if necessary (The revoke protocol is discussed in section 4.2.4)

The communication and major messages for this protocol are shown in Table 4.2.3.1. and Figure 4.2.3.1

Seq. No	Msg. Name	Sender	Receiver	Purpose
1	Get Order	Wallet	Vendor	Get regular order for Vendor
2	Confirm Order	Wallet	Wallet	Get user’s final confirmation to process order
3	Inquiry Payment	Wallet	Broker	Ask Broker to pay for the order
4	Request Order Delivery	Wallet	Vendor	Send Broker endorsed order back to Vendor
5	Register Order	Wallet	Host	Register processed order on System Host

Figure 4.2.3.1 Message Flow for Regular Payment Protocol

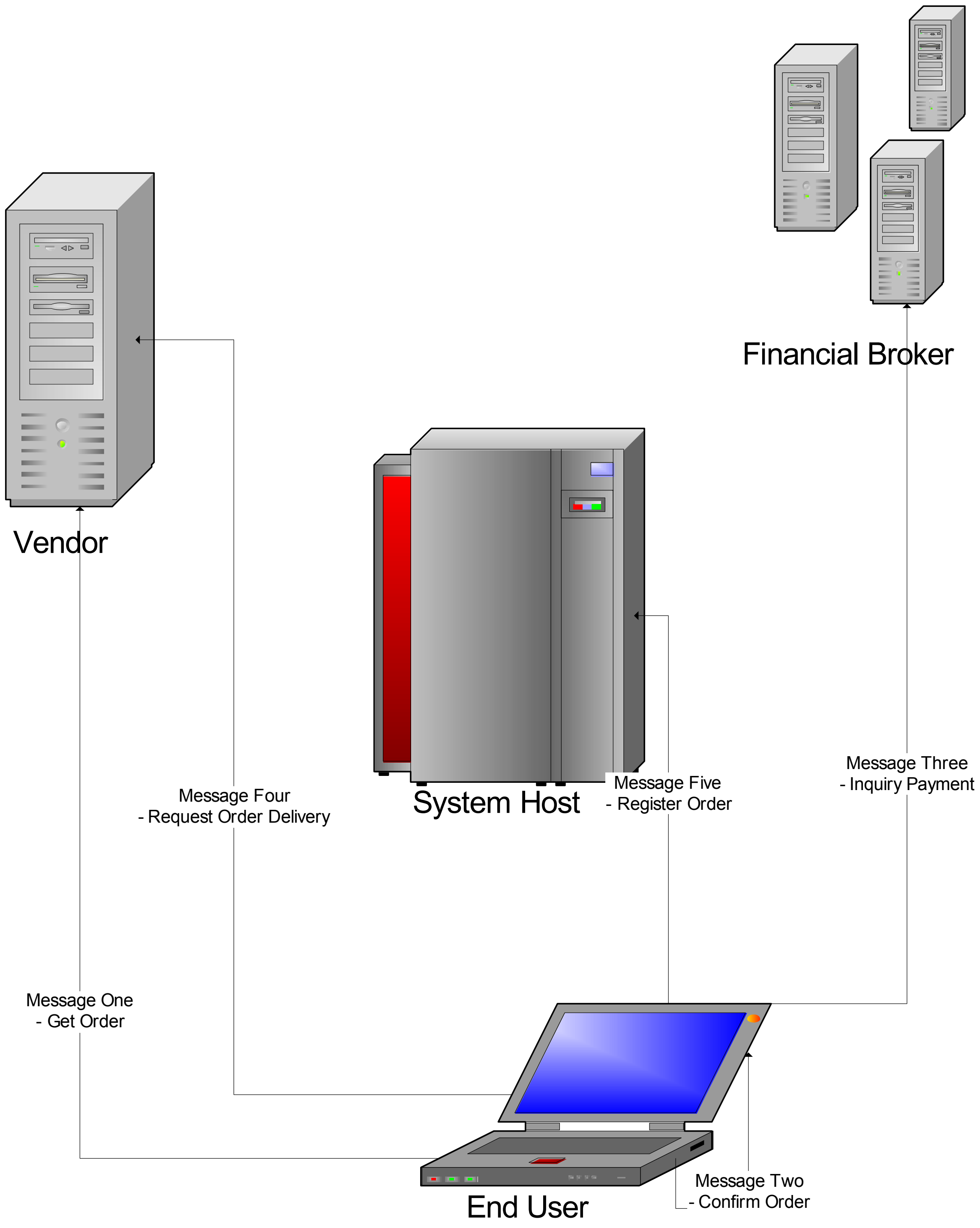

Financial Broker
Vendor
System Host
End User
Message One
- Get Order
Message Two
- Confirm Order
Message Three
- Inquiry Payment
Message Four
- Request Order Delivery
Message Five
- Register Order

Figure 4.3.3.1

4.3.4. Revoke Payment Protocol

Unlike micropayment, the regular payment is cancelable in QuickPay and Revoke payment protocol is used to cancel a committed payment. In the business world, a customer may want to cancel an order for some reasons. If Vendor accepts customer's request, it should refund customer's payment. The protocol here is for Vendor to refund End User's payment back and it should not be confused with a customer return. As a result, it is easy to see that it is Vendor instead of End User should invoke this protocol. With Revoke payment protocol in QuickPay, Vendor informs Financial Broker that it intends to cancel a committed order and won't claim payment for this order. (Sometimes, Financial Broker may already have paid for this order. In this case, QuickPay allows a Financial Broker to deduct payment from the next money transaction from Financial Broker to Vendor).

The major messages for this protocol are shown in Table 4.2.4.1.

Seq. No	Msg. Name	Sender	Receiver	Purpose
1	Revoke Order	Vendor	Broker	Inform Broker an existing order has been revoke
2	Register Revoke	Broker	Host	Register order revoke in System Host

Table 4.2.4.1 Messages for Revoke Payment Protocol

In this protocol, the underlying assumption is that Vendor is willing to revoke the payment, and does not have intention to abuse this protocol. It is assumed that Financial

Broker won't falsely charge its client for a revoked order intentionally. Once an order is revoked, neither System Host nor Financial Broker should charge a commission over it. Meanwhile, a major security risk for this protocol is that hacker may impersonate a Vendor to send a false revoke command. QuickPay has two ways to prevent this scenario. First of all, Financial Broker must check the IP address for the request and match it with the Vendor's IP address; Secondly, the Vendor is strongly suggested to check with System Host frequently in order to be able to get a false revoke alert earlier.

4.3. Communication Protocols

Communication Protocols are used for players to communicate with each other beyond a payment process or other application protocols. Generally, processes supported by Communication Protocols do not run on a regular basis. They run only when some special event occurs, such as sending updated information, exceptions, batch retrieval process call, etc. The data communicated with Communication Protocols are public information unless specifically mentioned. The communication defined in these protocols is one-way communication. It works as an asynchronous message pattern. The message producer first creates a message and registers it (on System Host) with one protocol, then the message consumers come to retrieve messages with another protocol. The message communication is not mandated. QuickPay always encourages relative parties to retrieve its message and act properly according to the message. But players must realize the unreliability of data, and should ensure their system does not rely on it.

Detail Description of Communication Protocols can be found in Appendix B.

5. QUICKPAY SECURITY AND PERFORMANCE ANALYSIS

As we discussed, the central issue faced by Online Payment systems is how to achieve optimal tradeoff between transaction costs and payment security. Unlike existing Online Payment applications, which mainly focus on developing effective algorithms to protect sensitive data with relatively low costs, QuickPay is designed to dramatically reduce the exposure of sensitive data during a payment process, and in turn, cut the security cost by eliminating the inherent security needs.

5.1. General Security Strategies

Generally, the security problems in online payment applications include user authentication (buyer and/or seller), data integrity (such as order price, total cost, etc.), and data confidentiality (such as credit card information). By dealing with such problems in different way, different online payment systems have their own strength and weakness (See Appendix A for detail discussion). QuickPay can be thought of as a hybrid of account-based and protocol-based systems. After carefully studying existing online payment systems, we designed QuickPay in the ways so that we can take their merits and avoid their drawbacks. These security strategies include:

Collateral verification. In existing payment system, the process is a linear. For example, in a token-based system, a customer buys tokens from a broker (or token issuer) and then pays them to a vendor. At the end, the vendor can be reimbursed for collected tokens from the Broker. In credit card payments, customer provides credit card number to a vendor and the vendor then uses such information to claim payment from the credit card issuer. In this pattern, the whole process works like a linear chain. If any step in this chain is broken, the whole chain is broken. For example, in token-based system, both customer (in step 1) and vendor (in step 2) could make forge tokens and break system. In order to reach an acceptable overall security standard, higher security measures must be implemented in each of the steps. Considering that an application has two steps, if the security risk for each step is 5%, then the overall risk could reach almost 10% as shown in following equation 1

$$10\% \approx 9.75\% = 1-(1-0.05)(1-0.05) \qquad (1)$$

In QuickPay, the workflow is paralleling instead of linear. For each payment request, Financial Broker gets it from both End User and Vendor. Vendor can only claim a payment from Financial Broker that has been registered by End User in the first place. With this pattern, the overall risk is diluted (instead of combined) by separate steps. In the example above, to achieve a 10% of security risk for the whole process, each step in QuickPay can afford a security risk as high as 34% as shown in equation 2.

$$10\% \approx 10.9\% = 0.33*0.33 \qquad (2)$$

In other words, QuickPay can tolerate a much looser security implementation (with much lower cost) for the payment transaction.

<u>Customer anonymity</u>. The "password problem" is one of the most critical security issues in account-based systems. An account-based system must use highly secure implementation to protect its user’s personal information during the payment transaction

and its own database. In QuickPay, all of the players except for Financial Broker do not have End User's personal and/or Financial Information and there is no need to exchange such information during payment transactions. As a result, QuickPay has no security needs and does not have to pay for user information protection. Regarding the Financial Broker, since it already has the End User's information for the other service it provides, the margin cost for QuickPay is, if any, very low.

Highly secure and automated user identification process. QuickPay separates the whole user identification into several steps. First, Customers can log in to Wallet with username and password. Unlike an account-based application, this log in is a local process and may not be needed at all. Secondly, Wallet identification is bonded with its IP address (or host machine's ID) instead of End User's login information. (Note: This identification is pretty much like identification with session id). Thirdly, Paycard identifies itself to its host Financial Broker with a secret Financial Broker-defined automated process. As a result, Financial Broker can use long-string key and/or secret and complicated algorithms to protect data integrity and confidentiality effectively.

Payment commitment per order. With credit card online payment, a customer needs to give his/her personal information to vendor. If such information is not handled properly or stolen by malicious credit card theft, it can be used to pay for a fraud order at the credit card owner's loss. In QuickPay, the Financial Broker's commitment over payment is absolutely based on the individual order. In other words, the Financial Broker's commitment cannot be abused in the same way as it is in credit card payment.

<u>Settling over argument by fair third-part</u>. In existing online payment transactions, each step involves only two parts. To ensure the transaction is valid and acceptable by two parties, either a special algorithm must be implemented (in token-based applications) or a trust relationship between them must be established in the first place (in account-based applications). As a result, transaction cost will be increased and/or user anonymity (one of the major advantages of online purchase) will be sacrificed. In QuickPay, System Host plays the role of a fair third party and silently watches whole transaction. As a result, QuickPay does not need special algorithms to ensure the transaction's validation and does not require both sides of a transaction to trust to each other. Since there are three parts in a transaction, QuickPay applies a so-called "majority vote" to verify a transaction's validation.

5.2. Security Challenges and QuickPay' s Solution

In section 4.1, we discussed general strategies used in QuickPay to deal with online payment security problems. In this section, we show how QuickPay uses these strategies to deal with typical security issues in online payment applications.

<u>Customer anonymity</u>. Obviously, a customer needs to identify himself/herself in credit card payments and account-based payment applications. Theoretically, token-based applications should not have any information about the user. But in practice, many token-based applications (such as PPay) do add some user-related information into the token in

order to trace its use history. As far as we know, there is no existing online payment application in which the customer is truly anonymous to all the players.

On the other hand, End User is totally anonymous in the scope of QuickPay. The End User in QuickPay is anonymous not only to the Vendor (like with most existing online payments) but also to the System Host. Of course, Financial Broker still has End Users (as its client) information. But this information exists beyond the scope of the QuickPay application. In other words, whether or not he joins QuickPay system, Financial Broker always has End User (as its client) personal/financial information.

<u>Identification stealing</u>. This problem is the most serious in credit card payment and other account-based payment applications. Theoretically, anybody who has valid user identification (such as a username/password) can impersonate that user and invoke a fraudulent payment transaction. As we discussed in section 4.1, QuickPay does not have such a problem and does not have to pay security costs on it at all.

<u>Fraud payment commitment and duplicate spending</u>. In token-based payment applications, the token (or coin) is regarded as a commitment from Broker. As a result, token-based applications are embedded with two serious security problems: token fraud and duplicate spending (Details have been discussed in the Part I Survey). Therefore, developing an effective algorithm to detect forged tokens and prevent duplication spending becomes the core part of almost all token-based payment applications.

With the design of Payment commitment per order, QuickPay can totally eliminate the duplicate spending problem because the Financial Broker's payment commitment is associated with a particular order. Meanwhile, with the design of Collateral Verification, a third party judge over argument, and other strategies, QuickPay can effectively detect and prevent forged payment commitment at low cost.

Overspending and credit risk. In online credit card payment and some credit type token-based or account-based applications, vendor or token-issuer allows customer to pay for an online purchase with credit and then collects cash from the customer later. As a result, there is a potential risk that the customer may overspend his/her credit and leave and the vendor or credit card or token issuer without final reimbursement.

In QuickPay, this risk can be dramatically reduced. First of all, the responsibility of reimbursement to Vendor is shifted to a Financial Broker that has a high reputation and financial reliability. As a result, Vendor's payment can almost be guaranteed. Secondly, the Financial Broker in QuickPay is supposed to have a long-term relationship with End User (as its clients), and knows their credit history very well. Thirdly, the Financial Broker's commitment is based on individual orders and can find any potential overspending instantly. Finally, the Financial Broker may also hold some financial assets (such as a saving account) for its customers, which may be used as collateral for the credit spending in QuickPay.

Denial-of-Service (DoS) attack. DoS attack is more significant in QuickPay because QuickPay is more open and communicates with plaintext format in most cases. Since the System Host's server is the most likely target for a DoS attack, the System Host should deal with this threat without cooperation of other players. QuickPay Protocol does not define System Host's implementation on this issue. In this way, the System Host can hide its defense algorithm and make it difficult for hackers to develop new types of DoS attacks.

5.3. QuickPay Performance Evaluation

QuickPay consists of several players. Among them, System Host is the center and the bottleneck of the whole QuickPay network and its performance is the most critical point of overall performance in the whole QuickPay system. Besides System Host, Financial Broker's performance is also very important in the QuickPay system but not as critical as System Host since there are multiple Financial Brokers in QuickPay system.

As a loose partner, Vendor is not the major concern in QuickPay's performance analysis. The basic performance concern of QuickPay to Vendor is to ensure that Vendor's operation should not become a block in the whole transaction process. Since Wallet and Paycard works on a client's machine, their performance should have only limited impact on the overall performance of QuickPay system.

Besides the different performance demands on different players, QuickPay also has different performance requirements for different Protocols. For example, Micropayment protocol is the most sensitive on performance while Administration protocols do not consider performance as one of the most important design issues. Therefore, we focus on micropayment protocols in following performance analysis (except that we explicitly mention other protocols and have the following distinguished designs on performance concerns:

<ins>Good overall performance</ins>. The transaction cost for online payment can be measured in two fields: the amount of online communication and the run cost of security operation over payment transaction. Account-based online payment application usually requires two online calls to System Host (get payment request from customer and send confirm notice to Vendor in PayPal) and may have some security operation costs (such as using SSL communication in PayPal). Token-based online payment application requires at least one online call to Vendor and a relatively big security operation cost (for token verification operation). In QuickPay, System Host and Financial Broker need to deal with three online messages respectively but neither of them needs to bear any security operation cost. As general knowledge, SSL requests require 3-5 times more computing time than no-SSL request while security hash algorithm (used for token verification in most token-based applications) can cost even more computing time. Therefore, as we can see, QuickPay should have similar, if not better, performance than other online applications.

<ins>No need for public key</ins>. Like many token-based and account-based applications, QuickPay does not use any Public Key operation in its micropayment transactions.

System host online support. Some legacy token-based applications (such as PayWord) do not use token Broker's online support and claim it as a big disadvantage. But with the dramatic reduction in cost of online communication, this concern fades. As a matter of fact, all account-based applications and latest token-based applications (such as PPay) require System Host (or Broker) to provide online support. In QuickPay, online support from System Host is required but can be temporarily suspended. .

Linear application load. Online application load is a measure of performance and can be regard as the amount of work an entity must do per unit time. In QuickPay, System Host's load obviously is directly related to the number of orders processed. Meanwhile, it should be a reasonable assumption that each Wallet (or each active Wallet) expectedly invokes the same number of orders in a certain period of time. Therefore, System Host's load is linearly related with the number of registered Wallet (or active Wallet) and System Host is highly scalable in support of Wallet.

On the other hand, the process orders in QuickPay should be shared by all of Financial Brokers. Therefore, if we have a reasonable number of Financial Brokers working in QuickPay (such as 100 of them), the load of each Financial Broker will be much lower than the System Host and should not be considered a bottleneck point for whole system.

System Host shares transaction burden Financial Broker. In many account-based and token-based online payment applications, a single site usually plays both the role of system administrator and Broker. As a result, that site bears major part of the system burden and constraints overall performance. QuickPay shifts a significant part of total performance burden from System Host (as single critical point) to multiple Financial Brokers and, as a result, obtains a better overall performance at a relatively low cost.

System Host works as a passive watcher. Unlike other applications, in which a host (or Broker) needs to be actively involved in a payment transaction and to do some heavy work, such as verifying customer authentication, committing a payment to a Vendor, etc. In QuickPay, System Host just silently receives the notices about a transaction from different players without any active involvement. As a result, the load of the System Host, as the bottleneck of whole system, can be much lighter and can handle much more online payment traffic potentially.

Some security work is shifted from regular payment transaction to batch process. Batch process improves the system overall performance in two parts. First, batch process can be run in some low traffic time so that the traffic on a System Host can be more balanced. Second, to process many orders together in one transaction is much more effective and much lower per unit cost than to do it for individual orders. For example, use of one signature against a thousand orders obviously is more effective and much cheaper than use of one signature against an order one thousand times.

Client-side components do the work as much as possible. As a general principle, QuickPay will have the client side component (Wallet and Paycard) do the work as much as possible. For example, QuickPay suggest Wallet's signature be verified by Paycard instead of its host Financial Broker.

5.4. Advantages of QuickPay

Based on above analysis of QuickPay's security and performance, we believe QuickPay can outperform existing online payment systems in following fields:

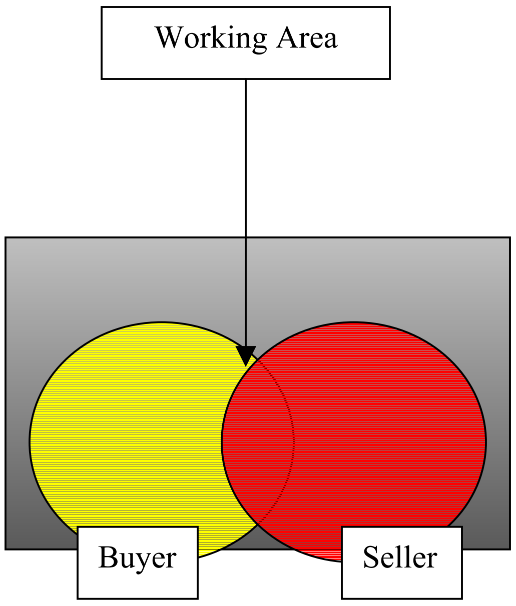
Simple support protocol. As discussed above, QuickPay does not imply any specific algorithms to ensure the online payment's transaction. As a matter of fact, QuickPay is purely built on HTTP protocol and RSA protocol for a public key. Since these protocols are standard W3C protocols and have already been supported by almost all the Internet servers and browsers and have been the standard protocols of all the web applications, QuickPay participants should easily be able to integrate QuickPay into their existing system without serious technical barriers.

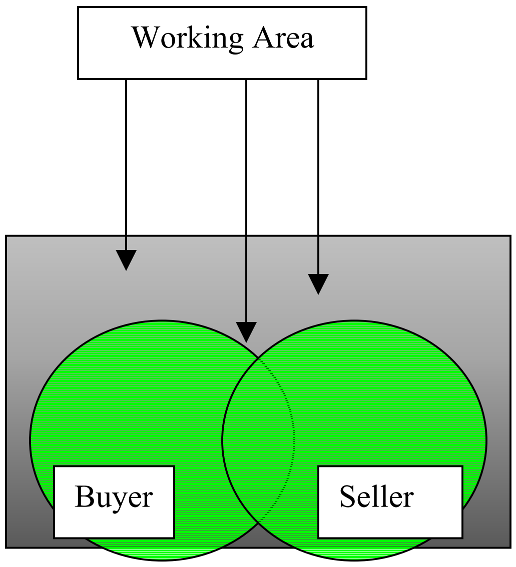
Technical support is minimal. QuickPay defines a set of protocols for communication among participants during an online payment process. Supporting the protocol is QuickPay's only technical requirement, and they are platform- and language-independent and also independent of each other. As a result, the implementation of QuickPay protocol can easily be developed as a standard package and integrated by QuickPay's participants.

High-scalability. In QuickPay, as mentioned earlier, the Vendor does not need to hold information on the End User, and vice versa. As a result, adding/removing a Vendor/End User is totally independent of other End Users/Vendors. Meanwhile, System Host does not need to be updated when Vendors/End Users update their own information since it does not hold such information. As a result, the whole system becomes highly scalable.

Openness**.** As we discussed, account-based payment system can only work for the transaction between its members. Technologically, such a system is closed and all the transactions for online payment actually occur “inside” a system environment. Contrary to this, QuickPay allows transaction over different Brokers. With this design, in our opinion, QuickPay will have a significant business advantage over an account-based online payment system. Here is a scenario about it (see Figure 5.4.1). For example, there is a successful account-based online payment system on which 50% of Internet users have registered. This system still can only cover 25% of online payments (the possibility of both sides of a transaction between that system’s members is 0.5*0.5=0.25). On the other hand, if two partners in QuickPay have only 30% of the market share individually, the ratio of total payment transactions covered by them can be up to 60% (=30%+30%). Therefore, QuickPay has the potential to support more online payment transactions with its open-ended design.

Common user GUI interface. As mentioned earlier, QuickPay has a standard client-side GUI interface for all the End Users, no matter which Financial Broker these End Users work with. With QuickPay standard client-side GUI, users can get the following major benefits: 1) users can learn and accept QuickPay easier and quicker; 2) a user can work with multiple Financial Brokers at the same time; and 3) a user can transfer from one Financial Broker to another without problem.





A. Account-Based Schemes B. QuickPay Scheme

Figure 5.4.1

<u>Job delegation.</u> Job Delegation is one of the major principles of a modern Object-Oriented Design. QuickPay takes advantage of this pattern by implementing it in the following fields:

- Besides delivering online payment transactions, many existing online payment systems, such as PayPal, PayBit, and others, also attempt to support whole E-Commerce process. The big advantage of this design is that the System Host can provide a whole package of services to its users but it also can make system more complicated and more costly. QuickPay focuses on the online payment transaction only. It has its partners (Vendors and Financial Brokers) to support other functionalities for E-Commerce based on their own circumstances. The underlying belief of this design is that these partners have the best understanding of their own customer and/or products, and have the best opportunities to build optimal services.

- In QuickPay's online payment transaction, the most critical part is the communication between End User and Financial Broker. With the use of Paycard, QuickPay has the Financial Broker, instead of System Host, to take care of the security for this communication.
- In account-based online payment systems, the System Host needs to manage a user's information. In QuickPay, this job is delegated to Financial Brokers too.
- Theoretically, QuickPay also won't guarantee an order's delivery. In other words, Vendors should take responsibility for their contents/products management and ensure a customer gets his/her order.

Special Concerns of micropayment and Instant Payment QuickPay has separate protocol for micropayment, which helps QuickPay to handle micropayment in more cost-effective ways as well as meet its special requirements, such as support for instant payment.

Extension. QuickPay is designed to consist of a set of independent protocol and each of the protocol is used to support a special online payment process. Since each of protocol is independent of each other, QuickPay can easily add a new protocol without interrupting with existing protocols. As a result, QuickPay has a big potential to be used for other online financial activities besides normal online payments. Here are some examples:

wPayment. Wireless Payment is becoming the hot issue currently. Since Wallet is associated with machines and there is no big difference between HTTP and WAP (or other wireless application protocols), QuickPay can naturally support Wireless Payment as long as we have a wireless version of Wallet (and Paycards).

Account Management and Transaction. Since QuickPay's Wallet can support multiple Paycards from different Financial Brokers, It should not be very difficult for QuickPay to support account aggregation and cross-institutional transaction via Paycards.

Online gaming or gambling. With instant content delivery and user anonymity design, QuickPay can become a perfect choice for those users who want to pay for online gaming or gambling without setting up an account there first.

Other online financial activities. Besides this, QuickPay also has the potential to be used for other online financial activities in the future, such as online ticket purchase for movies, online regular bill payment, online trade for stock or other securities, and so on.

6. CONCLUSION

In this project, we developed QuickPay, an efficient online payment application designed for different types of online payment and, potentially, other financial transactions. By identifying and focusing on the most common characteristics of different online payment situations, QuickPay can significantly outperform existing online payment applications. We are confident that QuickPay satisfies the core requirements to become a popular payment solution for a wide variety of online business activities.

REFERENCES

1. Anonymous, Case Study: PayPal, http://digitalenterprise.org/cases/paypal.html, September 30, 2004.

2. BitPass website. http://www.BitPass.com.

3. Chi, Ellis "Evaluation of micropayment Schemes" HP Laboratories Technical Report, n 97-14, pp 1-29, Jan, 1997.

4. Dai, Jian, “micropayment Token Schemes” Crossroads, , Association for Computing Machinery, 2004.

5. Hallam-Bakeer, Phillip, "Micro Payment Transfer Protocol", http://www.w3.org/TR/WD-mptp-951122, 1995.

6. IBM, "IBM Multi-Payment Framework (version 1.2)" http://www-306.ibm.com/software/genservers/commerce/payment/mpf.pdf, 1999.

7. KaZaA website. http://www.kazaa.com.

8. Lamport, Leslie "Password authentication with insecure communication" Communications of the ACM, Volume 24 No 11 pp 770-771, November, 1981.

9. Lesk, Michael "micropayment: An Idea Whose Time Has Passed Twice?" IEEE Security and Privacy, volune 2, No 1, pp 61-63, January/February, 2004.

10. Manasse, M. "The Millicent protocols for electronic commerce" proceedings of the 1st USENIX workshop on Electronic commerce, 1995.

11. MacKie-Mason, Jeffrey K. and White, Kimberly, "Evaluating and Selecting Digital Payment Mechanisms," in Interconnection and the Internet, G. Rosston and D. Waterman, eds. Lawrence Erlbaum, 1997: 113-134.

12. McCloud, Scott "Misunderstanding micropayment: BitPass, Shirky and The Good Idea that Refuses to Die", http://www.scottmccloud.com/home/essays/2003-09-micros/micros.html, September 11, 2003.

13. Micali, Silvio. and Rivest, Ronald L. "micropayment Revisited" CT-RSA 2002.

14. Milunovich, Steve. "micropayment's big potential" November 5, 2002 Red Herring.

15. Naick, Indran and Gupta, Reema, "Electronic payment processing for Web businesses", http://www-106.ibm.com/developerworks/ibm/library/i-money/, Feb, 2002.

16. Odlyzko, Andrew "The Case Against micropayment" Lecture Notes in Computer Science #1318, pp. 173, 1997.

17. Odlyzko, Andrew "Efficient micropayment system based on probabilistic polling" Lecture Notes in Computer Science #2742, pp. 77-83, Springer, 2003.

18. Palmer, Jonathan W. and Eriksen, Lars Bo "Digital newspapers explore marketing on the Internet" Communications of the ACM, Volume 42 No 9 pp 33-40, September 1999.

19. Patch, Kimberly and Smalley, Eric "Drop a Dime Online" http://www.infoworld.com/cgi-bin/displayStory.pl?/features/981130micropayment.htm, 1998.

20. PayCirecle website. http://www.paycircle.org.

21. PayPal website. http://www.paypal.com.

22. Pilioura, Thomi "Electronic Payment Systems on Open Computer Networks: A Survey" work paper http://wwwi.wu-wien.ac.at/public/ehandel/Zahlung_Ueberblick.pdf 1999.

23. Rivest, Ronald L. and Shamir, Adi "PayWord and MicroMint: Two simple micropayment systems" Lecture Notes Computer Science, 1318, pp 307-314, 1998.

24. Shirkey, Clay, "Fame vs Fortune: micropayment and Free Content", http://shirky.com/writings/fame_vs_fortune.html, September 5, 2003.

25. Siemens AG, and ICM N AS, "Identity management for Micro-Payments in a mobile environment", http://www.paycircle.org/downloads/01_download.htm, December, 2003.

26. Yang, Beverly and Garcia-Molina, Hector "Emerging applications: PPay: micropayment for peer-to-peer systems" Proceedings of the 10th ACM conference on Computer and communication security, October 2003.

27. Yen, S-M. "PayFair: a prepaid Internet micropayment system ensuring customer fairness" IEE Proceedings - Computers and Digital Techniques, Volume 148, No. 6, pp 207-213, November 2001.