Use of Currency for Access Control in Large-scale Information Systems

Ph.D. Thesis Proposal

Department of Computer Science
Columbia University

Apostolos Dailianas

September 1998

1. Introduction

Protecting large-scale information systems remains an elusive challenge of ever-growing importance and complexity. Exposure to insecurities and the opportunities available to attackers are increasing with the growth in the range of resources, scale, complexity, and operations management practices of domain administrations. Current information systems enable attackers to pursue virtually unlimited compromise attempts; they involve ad-hoc instrumentation to monitor resource access and manual correlation of access logs to detect intrusion; and they leave attackers unaccountable to abuses and crimes they commit. Rapid changes in technologies increase the vulnerability to attackers. First, at present protection technologies are component-specific. A minor insecurity in a new component can propagate substantial exposure to other components. A secure system can be rendered insecure by the addition of a single component. The combinatorics of interactions between new components and existing ones increase exponentially. Second, in the absence of a unifying security architecture it is impossible for component vendors, or domain administrations to accomplish a coordinated protection.

Domain administrations are thus increasingly exposed to security risks and are unable to control, bound or even assess this exposure. They require expert manual labor to monitor and correlate access anomalies and detect an attack, typically through off-line non-real-time processes completed hours or days after the attack has been completed. And even when an attack is detected, identifying the source accountable for it can be virtually impossible and requires complex ad-hoc collaborations of multiple expert police forces.

We propose the use of *currency* for access control as a novel approach to the protection of large-scale information systems. *Currency* is used both as a token of value and as a token of unique identification. Resources are instrumented to use currency for access control and monitoring. Resources include physical resources such as CPU cycles, storage, bandwidth, etc., as well as higher-level software services such as file storage, name service, databases, etc. Clients must pay resource managers, using appropriate currency and prices, to gain access to respective resources. Payment is performed through secure financial transaction protocols. Attackers are limited by their budget in gaining access to resources and in causing damage. Budgets are enforced by secure banks. Domain administrations control access to resources and establish quantifiable limits on exposure to attacks by adjusting prices of resources and controlling the availability of their currency.

Currency flows provide uniform resource-independent instrumentation to monitor and correlate access patterns and to detect anomalies. This enables the development of uniform resource-independent intrusion-detection mechanisms based on the statistics of currency flows. Intrusion-detection can be thus automated and accomplished in real-time with an attack. Furthermore, currency carries unique identifiers. A domain maintains full accountability of the entities to which currency has been allocated. A domain can account for sources of each access to its resources. In particular, once an attack has been identified a domain can establish verifiable proof of accountability in tracing its sources.

The proposed mechanisms are structured to admit scalability and enable protection among mutually distrustful domains organized in a large-scale federated system. These protection mechanisms are resource-independent and can thus be retrofitted into an existing system with minor adaptation of its components.

2. Related Work

Most of the work related to this thesis falls in one of the following four categories: (a) mechanisms for protection of individual components or whole network domains (e.g., firewalls, authentication, access control lists, etc); (b) electronic commerce mechanisms and protocols for secure trading over wide area networks; (c) economic-based mechanisms for network resource management; and (d) intrusion detection mechanisms.

A significant body of research and implementation work has been devoted in the past in protecting individual resources or whole network domains. Two of the most commonly used such mechanisms are firewalls/security gateways and a combination of authentication and access control lists. Firewalls ([5]) are typically computers that sit between an internal network and the rest of the world, filtering packets as they go by, according to various criteria. Even though their value should not be undermined, firewalls and security gateways offer limited security, slow down the system operation, and require special effort to support every possible new application that is developed. Authentication mechanisms ([15], [16], [24], [25]) establish the identity of an entity that wishes to access a resource, and access control lists (ACLs) determine the authorization of the entity to access a specific resource. An ACL is associated with each resource whose access needs to be restricted. ACLs become prohibitively expensive as they increase in size. They become expensive to store, hard to maintain, and provide little help in isolating attack sources once the source(s) of an attack has been identified.

Over the past decade significant research has been performed in the area of e-commerce, resulting in the development of several electronic payment protocols and respective financial institutions for secure transactions over wide area networks ([1], [4], [7], [8], [13], [22], [23], [26], [27], [28], [31], [32]). The financial institutions associated with this body of work do not address a few of the central issues in our work. The first is scalability. Most of the work focuses on centralized infrastructure particular to the associated payment mechanism. The second is the protection of the online financial infrastructure itself from intruders.

Also, in terms of the protocols for secure financial transactions, this body of work does not address some central requirements that arise from the application of such protocols in the architecture proposed here. Specifically, the volume of transactions created by trading both physical resources as well as higher level services is orders of magnitude bigger than what is assumed by typical e-cash protocols. Therefore protocols with very low overheads in terms of bandwidth, information that needs to be stored, and cost of the payment functionality are imperative. Another issue that is important is the provision of cheap guarantees that service contracts are honored. Service providers should deliver the requested services and customers must pay for the services requested and delivered. Finally, the nature of the traded commodities, sometimes referred to as “soft” commodities, is such that it cannot tolerate delays by the payment functionality.

Economic-based mechanisms for network resource management ([2], [6], [17], [20], [21], [29], [30], [35], [37]) have focused on the efficient allocation of resources through the application of economics-based principles and have provided insights on the role of prices and the operation of markets in a distributed network economy. We are building on this body of work by applying the accumulated knowledge to the creation of a market for trading access rights to resources and to the control of prices as a means to achieve access control and protection of resources.

Intrusion detection systems can be classified in two main categories: (a) *misuse detection* systems ([14]) attempt to identify intrusions by monitoring systems to identify patterns of well-known attacks; (b) *anomaly detection* systems ([19]) try to distinguish between normal and abnormal access patterns. Many intrusion detection systems currently involve manual ad hoc means; they rely on the experience of the mechanisms creators to invent thresholds for differentiating normal from abnormal behavior and employ hand-coded intrusion patterns in the attempt to identify intrusions. These practices limit their effectiveness and applicability to future unpredicted attacks. Furthermore, most of the intrusion detection instrumentation is resource specific, imposing restrictions on the correlation of events to detect abnormal access patterns. Current work ([18]) attempts to develop a systematic framework to semi-automate the process of building intrusion detection systems.

3. Use of Currency for Access Control

3.1 Objectives

The goal of this work is to develop novel information-systems protection mechanisms based on the use of currency and the application of market-based paradigms to access resources and services. These mechanisms seek to ensure the systematic, quantifiable and predictable survivability of large-scale information systems.

Specifically, the objectives of our research are:

- Provision of scalable, resource independent access to protected resources and services and establishment of quantifiable and tunable limits on the power of attackers to access or damage critical information systems resources
- Establishment of full accountability among separately administered and mutually distrustful domains; and provision for rapid tracing and isolation of attack sources
- Provision of resource-independent instrumentation to monitor resource access; detect intrusion attacks automatically, identify their sources and rapidly isolate attack sources and deny access
- Provision of quantifiable protection against loss of critical resources due to attacks or failures
- Efficient and graceful recovery from loss of resources through dynamic load distribution to alternate resources based on quantifiable priorities

3.2 Technical Approach

Resources and clients are organized in *currency domains*. Resources include physical resources such as CPU cycles, storage, bandwidth, I/O devices, or sensors as well as higher-level software services such as file storage, name service, database, or web service. A currency domain establishes access protection for a group of resources. It provides administrative infrastructure for imposing domain-level protection policies covering pricing of critical resources, assignment of budgets to internal clients, limitation of access by external domains, monitoring access to detect intrusion attacks and activating responses to attacks.

Domains issue their own currency. This currency is used to provide unified, scalable access to their services. The currency is uniquely identified by a *currency ID*. This establishes full accountability in the use of resources by tracing access to resources back to the holder of the currency. To gain access to the resources in a domain, clients first have to exchange currency of the target domain for their own. Currency of a domain gives the holder the right to access any of the resources in the domain, providing unified, scalable access. The currency of a domain encapsulates domain-level protection policies set by the domain. Specifically, domains control who can acquire their currency, along with the total currency outflow, the rate of currency outflow and other parameters, imposing strict domain-controlled limits on the access and attack power of any entity wishing to access the domain resources.

Currency domains encapsulate protection policies for the whole domain. Finer access control at the resource level is achieved through the pricing mechanism. Prices of resources along with available budgets of clients establish a dynamically tunable access control mechanism; provide the means for optimized load redistribution and graceful degradation upon loss; and impose quantifiable dynamically adjustable limits on the exposure to attackers.

Each resource in a domain is priced in terms of the currency acceptable by the domain. This price is advertised in respective service directories. Prices are dynamically updated to reflect various operation parameters such as access control policies and changing demand for a resource. The combination of prices and budgets available to clients provides a fine granularity, dynamically adjustable access control mechanism. Limiting access to a specific set of clients can be achieved by raising the prices to higher levels, guaranteeing that only qualified clients (those that have sufficient budget) can access them. Furthermore, currency identifiers enable additional price discrimination techniques. Budget and price discrimination can achieve a continuous spectrum of limits imposed on the use of a resource, based on the source domain of a request.

The pricing mechanism can also be used to reflect resource unavailability due to congestion or loss. A loss of a resource reduces the supply, thus automatically causing clients to redirect their demands to backup replicas. Reduced availability results in rising prices of the replicated resources. Rising prices create a natural selection process where applications automatically adapt to resource availability and obtain access to alternate resources according to their intrinsic priority captured by their budget. High-priority clients can apply their budget to continue and obtain high quality of service (QoS), while low-priority clients are priced-out. Thus a loss results in graceful selective degradation of services that optimizes the balance between available resources and demands.

Furthermore, prices can force the operation of resources within a "*desirable*" region of operation. The *desirable* region of operation is resource-dependent, and in general refers to the region of operation specified by the resource manager, where specific QoS constraints or other considerations are satisfied (e.g., the average incoming rate to a switch should be controlled to provide low delays and loss). Assume the purpose of attacking a resource is to move it to an "undesirable" region of operation. Then the price of the resource should reflect its reluctance to operate in that region. Should the attacker or coalition of attackers desire to sustain the attack, they would see a continuously increasing price to access the resource, forcing them to exhaust their budget at an increasing rate to sustain the attack. The pricing mechanism in this case provides a means to convert a "fixed" budget (belonging to a specific client or a coalition of clients potentially residing in different domains), to a much lower "effective" budget. Knowledge of the specific pricing policy can provide analytical upper bounds on the duration of attacks achievable by given collective budgets.

The power of attackers is limited by their available budget. An attacker can gain access to resources only to the extent that his budget permits it. Furthermore, with each access required for an attack, the remaining budget and with it the power of the attacker decreases.

Thus, *enforcement* of budgets (i.e., guarantees that no client or application can spend more than their budget) is a very powerful tool for limiting attacks and damages. Hierarchical budget enforcement is pursued by organizing network entities (i.e., resources, clients, or subdomains) in hierarchically nested domains; each domain has a bank that controls the budget usage of the entities inside the domain as well as the dissipation of currency to external domains. Bank servers maintain accountability among independent domains by controlling currency flows among them. The banking system is protected through strict traditional mechanisms (such as encryption and authentication) as well as market-based mechanisms (such as monitoring of currency flows, monitoring of exchange rates and balances, etc.).

Resource access is monitored to automatically detect intrusion attacks. Attack sources are identified and isolated. Monitoring is done in much the same fashion as in transaction processing systems. The unique currency identifiers are used to correlate resource accesses to sources of access. Currency flows provide a good way to model temporal behaviors of clients and patterns of resource access to classify activities into those that are legitimate and those that seem suspicious and hence warrant further inspection and authorization. Once an attack has been identified, identification through the currency identifiers isolates the source of the attack. This knowledge is rapidly propagated to other domains to avoid further spread of faults or attacks.

3.3 Novel Forms of Protection

Access to resources through currency provides several novel forms of protection. First, it provides *unified protection of access to critical resources*. Critical resources can be combined in a special currency domain. Access to these resources requires this special currency. External customers can convert their currencies to the currency of the protected domain. However, this conversion is strictly controlled and authenticated by the respective bank-servers of the domains, and limitations are imposed on the total currency of the protected domain that can be dissipated out. An attacker that took over another domain and breaks through the authentication of the bank servers still gains very limited access, subject to currency conversion limitations, entirely controlled by the protected domain.

Second, it provides *quantifiable and tunable limits on the power of attackers*. A first limit is imposed by the budget available to a client, which is in turn limited by the income it generates through the services it provides to the rest of the world. Spending within the available budget is enforced by the banking infrastructure. A second limit is imposed by the currency policies enforced by domains that have full control over currency dissipation to perspective clients. A third limit is imposed through price adjustment. Rising prices in case of attacks force attackers to exhaust their budget at an increasing rate to sustain the attack, thus limiting the effects of attacks. Knowledge of the specific pricing policy in this case can provide analytical upper bounds on the duration of attacks achievable by given collective budgets.

Third it provides a *uniform instrumentation for intrusion detection* and establishes *proof of liability*. Currency carries unique unforgeable identifiers that can be monitored and traced to establish full accountability of the source of an access; and provides uniform instrumentation to monitor and account for resource access in a give domain. Intrusion monitoring agents use this information to detect attack patterns and establish the domains responsible for it. Once an attack has been identified, identification through the currency identifiers isolates the source of the attack. This knowledge is rapidly propagated to other domains to avoid further spread of faults or attacks.

Finally, access to resources through currency provides a *unifying framework for the development of market-based resource management and protection mechanisms*.

Resource managers offer *protection of resource availability and graceful degradation under loss*. Critical resources are dynamically and rationally replicated to optimize expected revenues that they generate. Prices for resource access rise with increase in demand and fall with increase in supply. With a fixed supply, a rise in price indicates an increase in demand. Resource managers quantify the loss of revenue due to a loss of a resource, and hedge against it by investing current revenues to insure future availability. The services most valuable by their clients are provided with the highest redundancy. If supply does not increase, or decreases due to a failure or attack, higher prices moderate demand, limiting access to high-priority customers, i.e., those willing to pay the high prices and possessing the available budget. The rest are priced out and automatically redistribute their demand to similar services.

Customers deploy their own market-based *strategies for resource acquisition and protection against resource unavailability* that best satisfy their requirements under their budget constraints. Protection for resources that are critical for applications can be purchased to hedge against future resource unavailability due to failures or attacks. Customers dynamically quantify the importance of resources and services and rationally purchase resources or protection against resource unavailability in the form of rights to use alternate resources.

4. The Resource Access Layer (RAL) Infrastructure and Mechanisms

4.1 The RAL Architecture

We propose the introduction of a distributed protection middleware infrastructure, the Resource Access Layer (**RAL**), overlaid on existing infrastructure (Figure 1). RAL includes several mechanisms. Resource managers are responsible to set the price for a resource, collect payments for its access and deposit revenues with the bank server of the respective domain. Client managers are responsible to manage client budget, obtain pricing information and pass respective payments required to access services used by the client. Bank servers provide accounting, clearing and monitoring of currency flows. Price directories provide pricing information. These mechanisms are depicted in Figure 1 below.

In a typical scenario, depicted in Figure 1, a client belonging to domain X wishes to access a resource belonging to domain Y. The client needs to first obtain currency acceptable to domain Y. The client manager obtains respective pricing information and issues a request to the bank server of domain X to provide it with currency for domain Y. The bank server of domain X must obtain currency issued by domain Y and credit the account of domain Y with a respective central bank for this amount. The two domain bank servers pursue secure transactions with the central bank to accomplish this. Once the client manager obtains respective currency from its bank server, it can proceed to execute accesses to the service. Each access will incur a payment collected by the server manager.

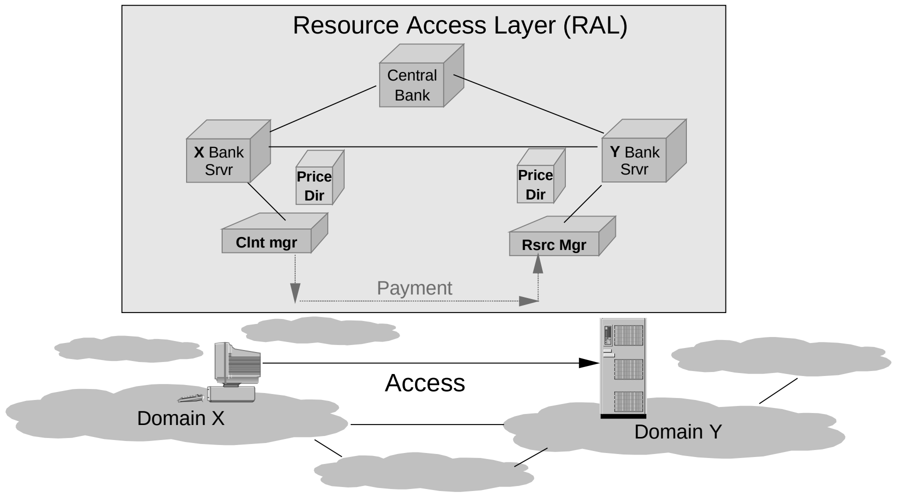


Figure 1 Overall System Architecture

It is important to note several salient features of this architecture.

- The RAL mechanisms provide incremental extensions of existing components and systems.
- The client and resource managers provide uniform (i.e., resource independent) and minimal extensions of existing software components to control access.

- All transactions involving currency flows between managers and their bank servers and between bank servers are secured through encryption and authentication.
- The overheads involved in converting currency among domains and in allocating currency to a client can be minimized through caching of currency. For example, the bank server of domain X can cache sufficient currency of domain Y in anticipation of requests by clients in its domain.
- Once a client obtains currency, the payment to resource managers involves very minimal overhead.

The RAL provides in effect a distributed secure access management kernel that is independent of underlying resources. In the following sections we present some of the essential components in implementing the middleware infrastructure depicted in Figure 1.

4.2 Currency Domains

The proposed architecture organizes resources (i.e., physical resources such as CPU cycles, storage, bandwidth, I/O devices, or sensors as well as higher-level software services such as file storage, name service, database) and clients in currency domains. Currency domains encapsulate domain specified protection of the included resources through strict control over currency dissipation; enable monitoring of currency flows, intrusion detection and tracing; establish accountability among mutually distrustful domains; and limit propagation of attacks.

The currency of a domain gives the holder the right to access resources in that domain. Therefore, to gain access to the resources in a domain, clients first have to use part of their budget and exchange it for the desired currency. Domain currencies are traded through secure domain banks at current exchange rates. The currency carries unique currency identifiers. Foreign domain banks requesting local currency are authenticated prior to the exchange and their identity along with the currency identifiers of the currency they acquire is recorded. Thus, the unique identifiers establish full accountability in the use of resources by tracing access to resources back to the holder of the currency. Exchange rates are set to balance the overall inter-domain access demands; they capture global behaviors and trigger alarms for abnormal access patterns. The currency carries unique currency identifiers.

A currency domain establishes access protection for a group of resources. It provides administrative infrastructure for imposing domain-level protection policies covering pricing of critical resources, assignment of budgets to internal clients, limitation of access by external domains, monitoring access to detect intrusion attacks and activating responses to attacks. Domains have full control over the dissipation of their currency. Specifically, domains control who can acquire their currency, along with the total currency outflow, the rate of currency outflow and other parameters, imposing strict domain-controlled limits on the access and attack power of any entity wishing to access the domain resources.

Domain currency establishes domain-level protection of resources. Finer access control at the resource level is achieved through the pricing and usage-monitoring mechanisms.

4.3 Banking Infrastructure

4.3.1 Introduction

The role of the banking infrastructure is to provide the means to guarantee quantifiable limits on the power of attackers. The power of attackers is limited by their available budget. An attacker can gain access to resources in a specific domain only to the extent that the currency of that specific domain he holds permits it. Furthermore, with each access required for an attack, the remaining budget and with it the power of the attacker decreases.

Thus, *enforcement* of budgets (i.e., guarantees that no client or application can spend more than their budget) is a very powerful tool for limiting attacks and damages. Budgets are enforced by the banking infrastructure. There are two aspects of budget enforcement we are considering. First, even if an attacker manages to legally accumulate or illegally produce big (or even unlimited) amounts of wealth, this should not compromise the security of other domains. Second, compromising any bank in the distributed banking infrastructure should not give the intruder access to unlimited amounts of currency.

4.3.2 Banking Architecture and Mechanisms

The banking infrastructure is based on the organization of network entities (i.e., resources, clients, or subdomains) in hierarchically nested domains. Each domain has a bank that monitors and controls the budget usage of the entities inside the domain. Each bank in the budget hierarchy monitors the budget usage of its children banks.

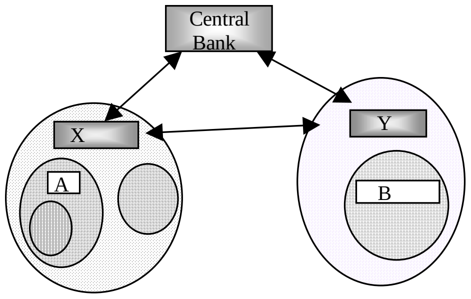


Figure 2: Banking Architecture

For example, in Figure 2, the budget of a user in domain **A** is controlled by the bank of this domain, and the budget of users in the entire domain **A** is monitored by the domain **X** bank. The role of the central bank is to control the generation of currency. Its role will become clear later in this section.

Limiting exposure to the wealth available to attackers

The primary means of limiting exposure of domains to external conquered domains is through the currency dissipation policies set by individual currency domains and enforced by the banking infrastructure. The total exposure of a domain to external attacks is limited by the total budget that it maintains in external domains and the rate at which it provides new currency to external domains. A domain can tune these parameters to control its exposure.

In Figure 2, a client from domain **A** that wishes to access services of **B** needs to pursue an exchange of local currency for currency acceptable by **B**. Exchanges of currency between domains are performed by the banks at the outermost enclosing domains. This restriction is imposed for scalability and authentication reasons. For example, a client from domain **A** that wishes to access services of **B** needs to pursue an exchange conducted between the **X** bank and **Y** bank. Domain **Y** bank has full control over how much of its currency it will exchange for currency offered by the **X** bank. This control is independent of the wealth available to clients in domain **X**. Therefore, the currency dissipation policies enforced by the **Y** bank can limit the exposure of the whole **Y** domain to external attackers independent of the amount of wealth they posses.

Further, organization of resources in currency domains – captured by the banking infrastructure – provides the means to scaleably limit the spread of faults or attacks and localizes their effects. For example, assume an intruder has conquered the whole **X** domain. Once any domain detects this, the information can be rapidly propagated to other domains and the currency of **X** can be declared invalid until appropriate action is taken to restore normal operation of the **X** domain.

Limiting the damage a conquered bank can cause

If we give a bank the ability to produce currency we cannot guarantee any limit on the internal domain wealth available to an intruder that has taken over the bank. A solution we are investigating is to give the ability to produce currency only to one bank – the central bank in the banking hierarchy. Rather than a single central bank, we envision the existence of a limited number of banking hierarchies, each with its own central bank. The inability of any bank other than the central bank to produce currency does not solve the problem in general, since one can still conquer the central bank and produce unlimited amounts of currency, but reduces its complexity to securing and strictly monitoring the operation of the central bank.

The operation of the scheme under investigation is based on the notion of *budget certificates*. Budget certificates are promises for future currency generation signed by the central bank and held at each bank in the hierarchy. The total amount of certificates in a bank node equals the wealth (budget) of the customers at that node, excluding the wealth of banks attached as children of that node. In Figure 2, a client from domain **A** that wishes to access services of **B**, sends the appropriate amount of budget certificates to the **X** bank. The **X** bank forwards the budget certificates to the central bank, requesting appropriate generation of **X** domain currency. This currency (carrying unique currency ids) is sent to the **Y** bank. The **Y** bank gets the appropriate **Y** domain currency from the central bank in exchange for the **X** domain bank it deposits. **Y** records the unique currency ids associated with this currency and the fact that this currency is given to **X**. The domain **Y** currency is now sent to the **X** bank. The **X** bank also records the transaction and associates it to the specific client that is the end recipient of the currency.

In this scheme, an intruder that conquers any node in the distributed banking hierarchy, does not have access to the budget available to entities (customers/banks) higher or lower in the hierarchy. This is because the intruder cannot produce the appropriate budget certificates required to request currency. The intruder only has access to the budget specified by the budget certificates stored locally. Further, the banks higher in the hierarchy monitor the intruder's use of these budget certificates. New budget certificates are created by the central bank and stored at respective banks when depositing takes place.

Mechanisms such as aggregation of budget certificates and caching of some amount of domain currency at the root bank of a domain (such as bank **X**), are investigated as means to guarantee that the banking infrastructure does not slow down the operation of the system.

4.4 Payment Protocols

4.4.1 Introduction

Access to resources and services through currency introduces the need for a protocol to securely transfer currency from the customer to the service provider.

Over the past decade significant research has been performed in the area of e-commerce, resulting in the development of several electronic payment protocols ([1], [4], [7], [8], [22], [23], [26], [27], [31]). Unfortunately this body of work does not address some central requirements that arise from the application of such protocols in the proposed architecture. First, the volume of transactions created by trading both physical resources as well as higher level services is orders of magnitude bigger than what is assumed by typical e-cash protocols. Therefore protocols with very low overheads in terms of bandwidth, very low storage requirements, and negligible cost of the payment functionality are imperative. Second, the need for very cheap guarantees that service contracts are honored, i.e., service providers provide the services they are paid for and clients pay for the services they are provided with, also becomes of central importance. Third, the nature of the traded commodities, sometimes referred to as “soft” commodities, is such that it cannot tolerate delays by the payment functionality. Therefore, the payment for resources has to be part of the transactions with minimal delays associated with it.

The approach proposed in the following sections provides secure payment functionality without delaying transactions. It has very low overheads in terms of bandwidth requirements and state information that needs to be kept by the entities participating in the trade. Further it provides an initial solution to the problem of decentralized policing of service provision. In the approach we propose, packets are carrying wallets paying for services they receive along the way. The customers keep paying as long they keep getting the service they requested, and providers keep providing service as long as they get paid for their services. This provides strong incentives for both customers and service providers not to misbehave and avoids the costly and time consuming policing functionality that would otherwise be required. Customers are policing service providers by discontinuing to pay if the service contract is not honored, and vice versa service providers stop providing service if the customer tries to cheat or does not pay in time.

4.4.2 Requirements

The payment protocol has to satisfy a number of requirements. First, since electronic cash is just a sequence of bits that can easily be duplicated, it has to guarantee that money is only used once. In other words, it has to guarantee that the customer does not double-spend the electronic currency and the service provider does not double deposit the same currency. A more relaxed version of this requirement, previously used in [4], [31], is that in case of double usage of money, some authority can indisputably distinguish whether the money was double-spent by the customer or double-deposited by the provider. This is the approach taken in the protocol proposed here.

Second, the cost of the payment infrastructure and operation should be negligible compared to the cost of the services themselves. This implies restrictions on (a) the amount of information transmitted as overhead in the transactions; (b) the amount of information that needs to be kept by the involved entities; and (c) the processing requirements.

Third, the delay associated with payment should be negligible. This implies that (a) the processing requirements are minimal; and (b) decisions about the validity of the received payment are taken locally without the need to contact remote authorities.

Fourth, currency should not be stolen along the way. In environments where a malicious user can anyway disrupt service by destroying packets, a reasonable relaxed version of this requirement is that stolen currency should be of no value to its new owner.

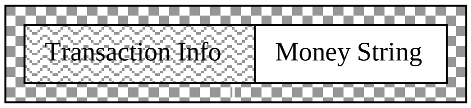
Fifth, it is desirable that the payment infrastructure provides guarantees that service contracts are honored, i.e., that service providers deliver the service they are paid for and that customers pay for the service they have received.

4.4.3 The proposed solution

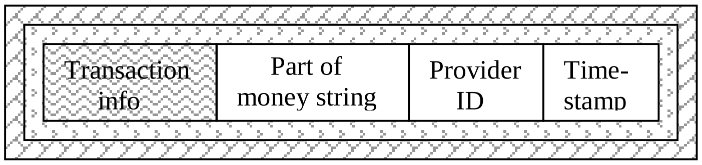
This section specifies the format and content of the messages exchanged between the different entities involved in a transaction. Section 4.4.4 describes how these packets achieve the desirable properties outlined in the previous section. The solution outlined here uses public/private key encryption ([15]). The public and private keys used for encrypting these messages are shown below:

Bank public	⬚	Bank private	▒	Shared secret key	▩
Customer private	⬚	Provider public	▨		

Bank-to-customer: the money withdrawn from the bank to pay for services is of the form depicted to the right. The *transaction info* field, which is encrypted with the bank's private key, contains a unique *money ID* and other information relevant to the transaction. It ensures the recipient that the specific bank is responsible for the validity of the money and provides other payment-related information. The *money string* is a simple binary string of some length associated with the *money ID*. Each bit of the *money string* represents some amount of money specified in the *transaction info* field. The reply from the bank to the customer is encrypted with a pre-established shared secret key corresponding to the customer.



Customer-to-service-provider: the withdrawn money is now used to pay for services. There is an important distinction between the *first payment packet* and *subsequent payment packets* for the same service. The **first payment packet**, encrypted with the customer's private key and the provider's public key, contains information for the whole duration of the transaction and is more space consuming and computationally complex due to the encryption. It has the form depicted to the left. The *part of money string* field contains the first few bits of the money string. The rest of the *money string* will be used in subsequent packets. Assume that the money string is: 01110100001100… In the first packet the customer might send the first, say, 7 bits of this string (01110100001100…). The next packet containing payment will start from the subsequent bits (01110100001100…). The *provider ID* uniquely identifies the provider the payment is intended for. The *timestamp* is simply the current time. The *<provider ID,*



timestamp> tuple will ensure no double-spending/depositing of money (as explained in the next section).

Subsequent payment packets simply contain a few more unencrypted bits of the money string. For example, in the above money string (01110100001100…) if we send the first seven bits in the first packet and if each subsequent payment header contains three more bits of the money string, we would next send 000 (01110100001100…), then 110 (01110100001100…), and so on. The provider slowly builds up the random string, which will be presented to the bank to get money credited on their account. If the provider presents to the bank only part of the random string, they will get credited accordingly.

4.4.4 Functional description of the protocol operation

This section describes the steps involved in the transaction and the actions performed by the involved entities.

1. The customer contacts the local bank and withdraws money to pay for the service.
2. The customer then sends the first packet containing payment information according to the protocol specification in the previous section. The provider decrypts the payment information with his private key and the appropriate fields with the customer's public key. The timestamp and the *provider ID* fields are now checked. If the timestamp is outside the time window for which information is kept in the provider's database (i.e., too old) or the *provider ID* is different than the provider's own ID, the payment is refused and the packet dropped. Otherwise, the provider decrypts the *transaction info* with the bank's public key to make sure the money is legitimate. A lookup for *money ID* in the local database is now performed. Notice that the provider keeps a record of only a few recent transactions within a time window dictated by the time skew between clocks, delays across the network and the storage space available at the provider's site. If the search succeeds (i.e., the money has been seen before), the payment is refused. Otherwise the *money ID* along with the timestamp is recorded. Service is now provided.
3. The customer keeps paying by sending more unencrypted bits of the money string as long as the providers keep providing the agreed upon service.
4. The provider takes the money to the bank either for checking during the transaction or to credit the money to his account at some time after the end of the transaction. The bank checks the *money ID*. If the money has not been deposited before (i.e., the *money ID* has not been seen before), and the *provider ID* is that of the provider depositing the money, it credits the provider's account and records the *money ID* along with the <*provider ID*, *timestamp*> tuple. If the *provider ID* is not that of the provider the bank refuses the transaction and the provider is liable for cheating. If the *money ID* has been seen before (i.e., the money has been deposited before), the previously recorded <*provider ID*, *timestamp*> tuple is compared against the received one. If they are the same the provider is cheating. Otherwise the customer is cheating.

4.4.5 Strengths and weaknesses of the proposed solution

The protocol has very low processing requirements, low overheads in terms of bits transmitted, and requires minimal state to be kept by providers and customers. It provides strong incentives for service providers to provide the service they are paid for through decentralized, real-time customer monitoring of service provision overcoming the need for centralized policing of service providers. Guarantees that money won't be used more than once, and that it can't be used for any purpose other than the intended (i.e., money that is useless if stolen), are provided through local, decentralized, off-line operations and decisions. The combination of these properties makes the proposed protocol a strong candidate for secure financial transactions in our framework.

We have identified a couple of weaknesses of the proposed solution, which require further investigation. The first is that the way the protocol was described, the only way for the service provider to know the received sequence is valid is to contact the bank and verify it. Expensive solutions to this weakness exist. For example, the first payment packet could also contain partial hashes of various parts of the payment sequence signed by the bank. The received payment sequence could then be compared against these partial hashes to check its validity. Apart from it being expensive, the partial hashes solution is also susceptible to off-line guessing attacks. The second is that an intruder can forge source and destination address and include a false part of the random number sequence, causing service disruption (since the service provider thinks the customer is cheating). Our current solution is to include a few verification bits along every payment packet. The verification bits are part of a verification string sent along with the first payment packet and their position in the payment packet is known in advance to the service provider.

4.4.6 Implementation

A prototype version of the protocol outlined above has been implemented in C. The protocol is used to conduct payment in the Msocket environment described in the next section. We plan to measure the performance of the protocol to prove its feasibility as a secure payment mechanism that does not significantly delay transactions. Initial experiments are encouraging.

4.5 The MarketNet Sockets (Msockets) layer

An important consideration for the adoption of the proposed middleware protection infrastructure is the ease of application development and porting. The initial strategy for its deployment and easy development of new applications and porting of existing applications is through the development of *Msockets*, a socket layer that mimics the syntax and operation of the Unix Sockets. Porting of existing applications is trivial. The operation of the initial implementation along with some of the software components developed to ease deployment of the protection middleware is depicted in Figure 3.

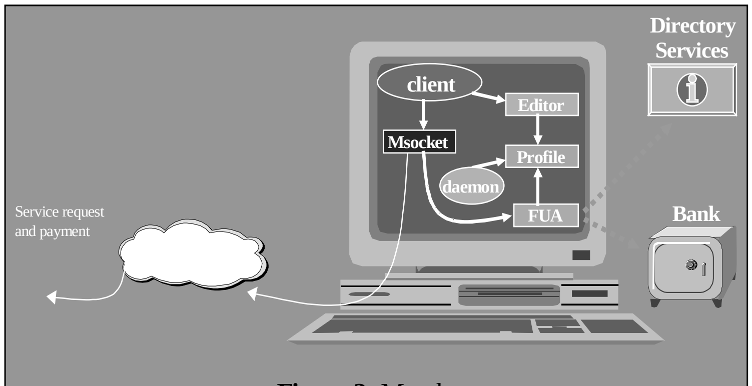


Figure 3: Msockets

Let us briefly describe what happens when a transaction takes place. Assume the client wants to access a service offered by a remote service provider :

- The client opens a *Msocket* to the server
- Before contacting the service provider, Msocket issues a call to the Financial User Agent (*FUA*), with the user's request. FUA is acting on behalf of the user; its main task is to automate budget management and make the currency infrastructure as transparent as possible to the user applications.
- FUA contacts the *directory service* and gets information related to thspecific service. This information consists of the cost of the service, and the currency the service provider accepts.
- FUA contacts the *user profile*. The *user profile* stores user preferences in the form "for service *x* spend no more than *y* over the period *z*". The profile is edited through the profile *editor* and automatically updated and reinitialized by the profile *daemon*.
- If the service is within budget, the budget information is updated, and the *bank* is contacted to acquire the appropriate currency for the transaction (local cashing of currency will in many cases eliminate this step).
- The currency is returned to the Msocket function that issued the call to FUA.
- The Msocket call establishes a connection with the service provider. The appropriate Msocket function at the provider side accepts the call, extracts the payment and starts providing the service.
- The Msocket function at the provider side, updates the server's profile, and periodically extracts payment sent by the client for the specific connection.
- At the end of the transaction, the Msocket at the client side instructs the FUA to deposit any unused part of the money withdrawn for the transaction. The FUA updates the user profile, and contacts the bank to deposit unused money.
- Similarly, at the provider side the FUA updates the earnings of the provider and deposits the revenues earned for the service provided.

4.6 Monitoring of Currency Flows for Intrusion Detection

Currency provides a uniform instrumentation to measure resource access and to identify anomalous patterns of access. Currency flows can be monitored and correlated in both space and time to discover potential attacks. Correlation in time enables monitoring of both temporal behaviors of resource accesses as well as the behaviors of source domains. The behaviors of source domains can be monitored both during currency acquisition as well as during resource access. Anomalies in access patterns can include elevated level of access by a particular source domain (e.g., an attacker accessing a critical service), or elevated level of correlated accesses by several source domains (e.g., an attacker using proxies). Correlation in space enables monitoring of correlated access activities to a group of related resources by a given source domain.

Instrumentation with currency enables use of uniform statistical correlation and anomaly detection algorithms to detect intrusion attacks involving arbitrary resources. This contrasts with specialized instrumentation and filters, and manual correlation presently used to handle the tasks. Furthermore, it permits correlation of access patterns to multiple related resources.

Once an attack has been identified, currency IDs are used to identify and isolate its source domain; the target domain can then automatically prevent any further accesses by the source domain by invalidating the respective currency, without affecting the operation of legitimate users. The target domain can resolve this potential attack with the source domain administration and these can identify and shut down access by the attacker.

5. Quantification of Attack Power

5.1 Introduction

A central goal of our work is the development of quantifiable bounds on the damage that can be caused by attackers. The aim is to describe these bounds mathematically (parametrically) to enable tuning of resource security. In this chapter we develop initial such bounds that quantify the effectiveness of the mechanisms; reveal dependencies among the security mechanisms; provide feedback for the design and operation of the security mechanisms; and capture the dependency of the mechanisms on the aggregate behavior of users (both attackers and legal users).

The security mechanisms that can be taken into account in this attempt are the following:

- Control of currency dissipation through currency domain policies
- Limits imposed by the banking infrastructure on the budget available to intruders
- Control of prices by service providers
- Intrusion detection and isolation of attack sources through monitoring of currency flows and resource utilization

5.2 Definition of Attacks

A major obstacle in trying to come up with bounds on attack power is that the definition of what constitutes an attack varies from resource to resource. For example, for a web server, a common attack is that of denial of service, where the attacker keeps bombarding the server with requests for service, essentially bringing the server down. In contrast, for a site supporting remote login of users, an attack could be the attempt of an attacker to login to the system by stealing or guessing a legal user’s password.

If an attack can be parameterized in terms of accessing patterns to the resource, then what constitutes an attack can be advertised as part of the access contract, and access compliance can be monitored and enforced. For example, part of the advertisement for access to a database could be that no client can pose more than, say, 10 requests per second. Unfortunately, such definitions are rarely possible. Furthermore, even though such an approach is useful in ruling out single attacking entities it fails in cases where a coalition of entities decide to collectively attack a specific resource.

From the multitude of cases of attacks, in our initial study we examine the case where we want to protect a resource from being forced to operate outside a *desirable region of operation*. The *desirable region of operation* is resource-dependent, and in general refers to the region of operation where specific QoS constraints or other considerations are satisfied. For example, consider any resource that can be modeled as a queue with a specific service rate. When the input rate of requests to the queue gets close to the service rate, or temporarily exceeds it, the queue will start growing and the requests will see an ever increasing delay for service; eventually some of them will be dropped. Therefore the service provider could consider the *desirable region of operation* to be that where the total demand from customers does not exceed, say, 80% of the service rate.

5.3 Parameters

In this section we introduce the parameters we are going to use for quantifying the power of attackers. We are currently examining a subset of the parameters defining the system operation. The subset refers to the dissipation of currency performed according to domain policies and the role of dynamic prices for access to resources and services. In the future, the parameters referring to the limits imposed by the banking-infrastructure protection mechanisms, along with the limits introduced by the transaction monitoring mechanisms will be introduced in the equations.

Let the following parameters describe the currency dissipation policies of a domain:

- A_{tot} – maximum total amount given to external domains
- A_{tot_dom} – maximum amount given to a specific domain
- R_{tot} – maximum total rate of currency outflow
- R_{tot_dom} – rate of currency outflow to a specific domain

Also, let the price per unit of resource or service per unit of time be denoted by

- $P(c)$

where c is the total demand for the service. Therefore, a user with demand c' will pay c'P(c) per unit of time. The function P(c) generally reflects availability of the resource and other considerations such as the willingness of the provider to restrict operation of a resource within the "desirable region of operation". P(c) may not be continuous (e.g., it could be a step function). For now we assume that the price for the service only depends on the demand and is independent of time. Also for now we assume that there is no price differentiation, i.e., two customers with the same request will see the same price. Finally, we assume that the customers see a price per unit that is independent of the units of service they request. All these simplifying assumptions in terms of pricing, are worst-case assumptions in terms of protection of resources.

5.4 Bounds

This section presents two examples of bounds that can be derived from the above parameters and mechanisms.

5.4.1 Damage that can be caused to a specific service by a given budget.

The budget considered here belongs to a single attacker or a coalition of attackers. A central *assumption* is that the attackers have a *fixed* budget to spend on the attack, i.e., they do not acquire more domain currency during the attack. This is the simplest case since it does not depend on the rate of currency dissipation.

Assume an attacker (or coalition of attackers) wants to bring a resource to an "undesirable region" of operation. For the purposes of our analysis, the attacker wants to increase the demand for this resource to such a level that other users will see a degradation in the quality of service they observe. Call this level C_{thres}. This level is defined by the service provider with respect to considerations about the QoS offered to customers. The worst case for the attacker (or coalition of attackers) is when no other user is using the service, therefore the attacker has to incur the cost for a demand level equal to C_{thres}. The best case for the attacker is when the demand of other users is very close to C_{thres}. In this case with minimal demand (and therefore cost) he will bring the system in the undesirable region of operation. Even when the system starts in the best case for the attacker, it will gradually move to the worst case, since other users experiencing congestion coupled with higher prices for the service, will redirect their requests to other service providers. We further assume that the service is critical for a number of customers with collective demand $\boldsymbol{C_{crit}}$. These customers will still demand service from the service provider ignoring the rise in prices or the congestion they are experiencing. The pricing model has to guarantee that the critical demand can always be accommodated; for example by rising the price to extraordinary amounts when C_{max}- C_{crit} is reached, where $\boldsymbol{C_{max}}$ is the maximum service capacity for service of the resource.

Let $\boldsymbol{C_{others}(t)}$ denote the capacity requested collectively by legal users at time t. Then for the attacker to bring the system to the undesirable region of operation incurring minimum cost, the following should hold at any time t:

$$C_{others}(t) + C_{attacker}(t) = C_{thres} \quad \text{or } C_{attacker}(t) = C_{thres} - C_{others}(t).$$

It follows that the duration of the attack, $\boldsymbol{T_{attack}}$, depends on the collective behavior of the other users (i.e., how fast their collective demand will reduce to C_{crit}). Let $\boldsymbol{B_{attack}}$ be the total budget available for the attack. The attack duration as a function of the collective behavior of other users, is the solution for T_{attack} of the following equation:

$$\int_0^{Tattack} C_{attacker}(t)*P(C_{thres})\, dt = B_{attack} \Rightarrow \int_0^{Tattack} (C_{thres} - C_{others}(t))\, dt = B_{attack} /P(C_{thres})$$

$$\Rightarrow \int_0^{Tattack} C_{others}(t)\, dt = T_{attack}*C_{thres} - B_{attack} /P(C_{thres})$$

In the worst case the budget B_{attack} available for the attack, equals A_{tot}. This upper limit does not make much sense, since it would prevent other users from accessing the service, but nonetheless it gives an upper bound on the value of B_{attack}.

There are a few interesting things to notice from the above equation. First, the price $P(C)$ of the service for congestion levels below the threshold C_{thres} does not affect the worst case duration of the attack. Second, the collective behavior of other users is critical to the duration of the attack. The sooner they stop requesting service, the sooner the attack will terminate. This favors implementations where a user agent adjust its access patterns at the provider site. Third, assuming the price for a given service is independent of time (as in the formula above, where it is only a function of the congestion level), the parameter that effects the worst case duration of the attack is not B_{attack} but $B_{attack} /P(C_{thres})$. This constitutes an "*effective attack budget*". A time-dependent pricing policy that rises the price for a given congestion level with respect to the time the resource stays in a congestion level would lead to a steeper decline of $C_{others}(t)$. Finally, the equation demonstrates the potential usefulness of price differentiation.

5.4.2 Damage that can be caused to a specific service.

This is similar to the previous case, only here we relax the assumption that the budget is fixed. We assume that the attacker or coalition of attackers can keep acquiring domain currency while the attack is in progress.

Assume that the attackers start with an initial budget $\boldsymbol{B_{attack\_init}}$ for the attack and during the attack they keep acquiring domain currency at the maximum allowable rate $R(t)$. Similar to the equation in the previous case, we can write:

$$\int_0^{T_{attack}} C_{attacker}(t)*P(C_{thres})\, dt = B_{attack\_init} + \int_0^{T_{attack}} R(t)\, dt$$

$$\Rightarrow \int_0^{T_{attack}} (C_{thres} - C_{others}(t))\, dt = B_{attack\_init} / P(C_{thres}) + \int_0^{T_{attack}} R(t)/ P(C_{thres})\, dt$$

$$\Rightarrow \int_0^{T_{attack}} (C_{others}(t) + R(t)/ P(C_{thres}))\, dt = T_{attack}*C_{thres} - B_{attack\_init} / P(C_{thres})$$

For the above equations to have a solution for T_{attack}, i.e., for the attack time to be bounded, we need to guarantee that the attacker spends money on the attack at a higher rate than the rate at which more money is acquired (this is apparent from the first form of the above equation). This means that after some point in time, T', after the attack has started, we need to guarantee that $C_{attacker}(t)*P(C_{thres}) > R(t)$. Assuming that by time T' the request by other users has fallen to C_{crit}, this restriction can be rewritten as $(C_{thress} - C_{crit})*P(C_{thres}) > R(t)$, which gives as an indication of what the price should look like beyond the threshold to be able to guarantee that the attack will have a limited duration even under worst case conditions.

The above equation is pessimistic in the sense that it assumes the intrusion detection mechanisms will not kick in to stop the attack when they see a request for domain currency equal to the maximum dissipation rate $R(t)$ for a prolonged period of time.

5.4.3 Other bounds

Similar rational, along with extension of the parameters that are taken into account, can be used to derive bounds on other aspects of the system operation, such as:

- The time to detect an attack
- The amount of currency a single compromised domain can acquire from another domain before its compromise or intention to attack are detected.
- The amount of currency a single compromised domain can acquire from a collection of other domains before its compromise or intention to attack are detected.

5.5 Other types of attacks – The Worm Attack

In the previous sections we have examined a particular kind of attack. The results that can be derived for such attacks may not be of much help in other kinds of attacks. Hence, further investigation and research is needed. In this section we take as an attack example the famous "worm attack" ([33]) and show the kind of protection and bounds the proposed currency-based access mechanisms offer.

The purpose of the worm was to spread itself, i.e., to break into systems, install and locally run itself. To achieve this goal it exploited flaws in utility programs based on BSD-derived versions of Unix. The worm program eventually spread itself to thousands of machines around the Internet, and disrupted normal activities and Internet connectivity for many days. The disruption was caused by the worm's heavy use of system resources in order to break to other systems. The worm was not destructive in nature, i.e., it did not delete user files or try to cause other such disruptions, but it has to be noticed that if its intention were to do so, nothing would have prevented it.

The worm spread using three techniques:

1. Exploiting a bug in the *gets* standard C I/O library used by the finger daemon (*fingerd*). The *gets* call takes input to a buffer without doing any bounds checking. The worm would overran the buffer and rewrite the stack frame, allowing it to gain control in the target machine and install and execute itself locally.
2. Using the *debug* option in *sendmail* servicing the SMTP port. This feature enables testers to run programs on remote machines to display the state of the remote mail system without sending mail or establishing a login connection.
3. Guessing passwords and exploiting trust between hosts. Once present in a system the worm would try to guess user passwords, break into their accounts and exploit trust between hosts (e.g., hosts found in /etc/hosts.equiv and /.rhosts) to remotely execute into those systems as well. Password guessing is a computationally very intensive operation using a big portion of the system resources.

Using any of the above methods, the worm would successfully spread without leaving any trace of where it came from.

Assuming the proposed security infrastructure were in place:

1. The prospective attacker would have to pay to use *sendmail* or *fingerd*, leaving an unforgeable trace of the originator of the attack.
2. Using system resources is not free. To perform password guessing the process would involve heavy system resource utilization. Monitoring of the budget usage at the conquered account domain would soon trigger alarms due to the unusual behavior. Furthermore, the amount of damage (e.g., overloading system resources) the process can achieve is limited by the budget available to it. Notice that we make a worst-case assumption, namely that the intruder manages to use the budget available to the account for using the system resources.

The worm attack is one of the most difficult attacks to handle and shows some of the limitations of the proposed system. These limitations are not particular to currency-based access control. They stem form the fact that software implementation bugs may allow intruders to impersonate legal users gaining the same privileges of legal users. We are currently investigating how the following scenarios can be efficiently handled:

Assume that the worm had destructive intentions. Budget enforcement along with usage monitoring would limit the scope and the extent of the damage. Price discrimination techniques may alleviate the effects of such attacks by making resources very expensive when the process does not normally use them.

In a worm-like attack, the attacker manages to impersonate the owner of an account. Even when this happens, it should not be equivalent to getting hold of the budget available to the account. One of the mechanisms to break this equivalence is usage monitoring. Abnormal access patterns can be restricted providing adjustable limits on the amount of damage a malicious or faulty processes can cause. A second mechanism under investigation is the separation of budgets available on a per process and/or per task basis. The tradeoff in this case is protection level vs. ease of use of the system.

6. Hedging to Reduce Uncertainties Due to Loss or Congestion

6.1 Introduction

The use of currency for access control in large-scale information systems creates a framework that enables the development of new market-based client and resource managers for resource acquisition, management, protection, rationalized replication, introduction of new services, and others. As an example of such mechanisms, in this chapter we develop a model for the provision of cheap, versatile, user-defined quality of service (QoS) guarantees. The model is based on the use of financial instruments to protect against resource unavailability due to loss or congestion.

Financial markets have created risk reduction instruments such as futures and options, used to purchase resources in the future and hedge against intrinsic uncertainties in the price and the availability of resources. Analogous instruments can provide QoS guarantees in network environments. Resources such as bandwidth are traded through the use of financial instruments. Applications try to optimize the QoS they get, subject to their budget, by purchasing instruments that entitle them to use resources in the future. Similarly service providers can rationally purchase backup resources or replicate their own resources to optimally guarantee continuous provision of services to their customers.

To demonstrate the concepts, we study QoS guarantees that depend on bandwidth demands. Bandwidth is traded through the use of financial instruments similar to those in financial markets. We are using two types of instruments: forwards contracts and options on forwards contracts. The mechanisms presented are not restricted to these instruments. The participants in the market are the service provider and the customers. Both can assume the role of buyers or sellers. Offers and requests along with the corresponding prices are posted in a bulletin board, which plays the role of a broker that matches the requests and the offers. Two kinds of customers are considered: customers with strict QoS guarantees and high budgets and customers with loose QoS guarantees and low budgets. The former purchase guaranteed bandwidth in the form of forwards contracts. The rest try to achieve their QoS requirements through the use of cheap best-effort service, and purchase financial instruments to hedge against the uncertainty associated with competing demand for best-effort service. Customers interested in buying guaranteed service participate in the market as buyers, while customers who have previously purchased guaranteed service they are not going to use participate as re-sellers. Examples of their strategies for trading instruments are presented in Section 6.3.1. Prices are dynamic, driven by demand and by competition among the service provider and the customers reselling previously acquired resource contracts.

The system provides differentiated, user-defined QoS guarantees, and can support new QoS guarantees without the need for new hardware or software support. Users get the QoS guarantees they want – they are not restricted to one of the QoS classes traditionally offered by systems providing QoS guarantees. Better resource utilization and cost reduction can potentially be achieved, since users purchase only the resources they need. Finally, users see a system where they can accurately tailor the QoS they get to their available budget.

6.2 Forwards contracts, options on forwards contracts, and markets

Two kinds of instruments, forwards and options, are used for the demonstration of the protection mechanisms developed. The mechanisms we present are not limited to these particular kinds of instruments. [3] and [12] provide thorough coverage of instruments.

Forwards Contracts. A *forwards* contract is an agreement between two parties to buy or sell an asset at a certain time in the future for a certain price. The future time the contract refers to is called *maturity time*. There are two parties involved in every forwards contract. One party takes the *long position* and buys the forwards contract. The other party takes a *short position* by selling the forwards contract. A range of commodities and financial assets can form the underlying asset in a contract (e.g., sugar, cement, stock indices, treasury bonds, etc.). The forwards price of some commodities increases as the time to maturity increases (*normal market*), whereas the price of others is a decreasing function of the time to maturity (*inverted market*). As the delivery time of a forwards contract approaches, the forwards price converges to the spot price of the underlying asset (*convergence property*). Forwards are extensively used to hedge against price uncertainties.

Options Contracts. In forwards contracts, the holder is obligated to buy or sell the underlying asset. In contrast, an option contract gives the holder the right to do something. The holder does not have to exercise this right. Whereas it costs nothing to enter in a futures contract, there is a cost to entering into an option contract. The underlying assets include stocks, stock indices, foreign currencies, debt instruments, commodities, and forwards contracts. There are two basic types of options. A *call options* gives the holder the right to buy the underlying asset by a certain date for a certain price. A *put option* gives the holder the right to sell the underlying asset by a certain date for a certain price. The price in the contract is the *exercise* or *strike price*. The date in the contract is known as the *expiration date, exercise date* or *maturity*. There are two main types of options contracts. *American options* can be exercised at any time up to the expiration date. *European options* on the other hand can be exercised only at the expiration date itself. In this work we consider european options. Similar to forwards contracts, one of the sides assumes the *long position* (buyer) and the other assumes the *short position* (seller). The price of an options contract is based on the price of the underlying asset. It is typically calculated through the Black-Scholes formula.

Markets. Trading of forwards and options contracts takes place in two kinds of markets. In the **primary market** new issues of forwards contracts are offered to the customers. In the **secondary market** trading among investors of already issued securities takes place. Trading in the secondary market does not affect the outstanding amount of securities; ownership is simply transferred between investors.

6.3 Application to the domain of bandwidth reservation

We demonstrate the application of the proposed mechanisms to bandwidth reservation. Time is divided in time slots. Within each time slot, there is a specific number of bandwidth units available. Part of the bandwidth within each time slot is sold as guaranteed service through forwards contracts by the service provider, and the rest as best effort service. There is a single service provider (for simplicity), and several customers. Each customer can have its own strategy for trading bandwidth. Each participant tries to maximize their utility function subject to their budget constraint. The service provider wants to maximize revenue and guarantee that the maximum number of users are satisfied; users want to satisfy their QoS requirements incurring the minimum cost. For simplicity in the presentation we assume that there are two kinds of customers. *Rich* customers that have a high budget and request strict QoS guarantees, and *poor* customers that have a limited budget and request the best QoS guarantees their budget can afford.

6.3.1 Example strategies

The example strategies presented here are consistent with the selfish optimization objectives of the participants. They are not unique. One could easily experiment with alternative strategies or experiment with other markets, such as non-monopolistic or economies involving speculators. Experimenting with different strategies provides a flexible way to implement new resource management schemes**.**

Service provider – Selling strategy

The service provider sells forwards contracts on the reserved part of the bandwidth. His goals are to maximize revenue and guarantee the proper operation of system resources. We assume proper operation of system resources to mean maximum utilization of bandwidth. The provider's goals are reflected in the prices advertised for units of reserved bandwidth in future time slots. The dynamic pricing scheme, presented in Section 6.3.2, encapsulates the strategy of the service provider. The *assumption* for the pricing strategy presented in this paper is that the service provider is not taking advantage of the monopolistic nature of the market. We assume for simplicity that the service provider does not participate in the secondary market.

Service provider – Buying strategy

The service provider does not buy bandwidth. Alternative strategies where the provider participates in the secondary market as a buyer, could target tighter control of the market, influence of the market prices to achieve desired system operation (e.g., traffic smoothing), control over the behavior of users trading bandwidth in the secondary market, etc.

Rich customers – Buying strategy

Rich customers try to acquire the required bandwidth to strictly guarantee that the QoS requirements of the traffic will be met. They are always willing to pay the cost for acquiring the bandwidth at the market price set by the service provider. They are also always willing to buy enough bandwidth to absorb any uncertainty in their traffic characteristics. They buy as early and as far in the future as allowed by their knowledge of traffic characteristics and the policy of the service provider. For example, if they know the peak and the mean rate of their traffic, they will buy forwards contracts from the service provider for the peak rate for the whole duration of the connection. For simplicity in implementing the bandwidth purchase decision algorithm, rich customers in our example do not buy reserved bandwidth from the secondary market.

Rich customers – Selling strategy

Rich customers try to minimize the loss of budget due to purchase of unneeded bandwidth. They try to sell unneeded bandwidth, as soon as they realize they will not use it. They try to sell it in the secondary market in the form of options on the forwards contracts. The price of the options is calculated through the Black-Scholes formula.

Figure 4 depicts the example buying and selling strategy outlined above. The rich customer knows at time t_{start} that the peak rate is 7 bandwidth units per slot (bups), the mean rate is .75 bups and the call will last until t_{end}. The customer does not know the exact traffic pattern, so he buys 7 bups in all slots from t_{start} to t_{end}. Assume that the traffic is bursty and that the customer knows at the end of each burst the starting time of the next burst. Then, at t_1 he knows the next burst will begin at t_2 and so on. At t_1 (i.e., as soon as he knows) he goes to the secondary market and puts the bandwidth up to t_2 for sale in the form of options. The same happens at t_3 (for the bw up to t_4) and at t_5 (for the bw up to t_{end}). There is no guarantee that there will be buyers for these options in the secondary market.

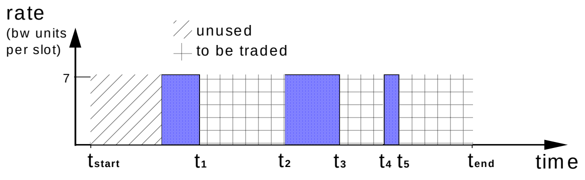


Figure 4

Poor customers – Buying strategy

Poor customers try to purchase cheap guarantees that their QoS requirements can be met under a restricted budget. Typically they try to send their traffic using best-effort service which is cheaper than guaranteed service. They only buy call options to use guaranteed bandwidth when they predict there will not be enough best-effort bandwidth to guarantee their QoS constraints. They prefer options that maximize their flexibility in deciding whether they will need to exercise the options or not. These are options for slots as close as possible to the transmission deadline of the traffic they try to protect, and as close to the underlying time slot as possible.

Poor customers – Selling strategy

Poor customers try to minimize losses due to purchase of unneeded bandwidth. They try to sell call options as soon as they realize they are not going to exercise them.

Figure 5 depicts an example of a buying strategy for a poor customer. Assume that at time t_{start} the customer produces a burst. The burst can tolerate a maximum delay (D_{max}), i.e., if not transmitted by $t_{start}+D_{max}$ it is considered lost. At time t_{start} the customer buys an option to purchase guaranteed bandwidth at $t_{start}+D_{max}$. The option expires at $t_{option\ expiration}$, i.e., by that time the customer has to decide whether he is going to exercise the option or not. If by the time the option expires he has managed to transmit the burst through best effort, he will try to resell the option at the secondary market. If not, he exercises the option to buy the guaranteed bandwidth. In the worst case the customer has overpaid an amount equal to the premium paid for the option.

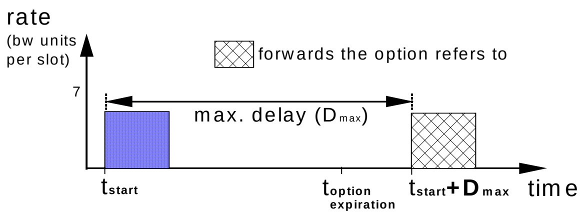


Figure 5

6.3.2 Pricing of bandwidth – The prices of forwards and options

The pricing policy encapsulates the strategy and dynamically reflects the goals of the service provider. In our example the goal of the provider is to maximize utilization of bandwidth. This translates to giving customers incentives to prefer low-demand bandwidth slots over high-demand ones. This is reflected in the pricing policy that initially assigns prices as a function of the demand for the same time slots in previous days (projected demand based on statistical data for the corresponding time slots in the past). Prices are dynamically adjusted to reflect changes in the current (real) demand. An increase in the real demand leads to a corresponding increase in the price of the bandwidth, encouraging users to distribute their load to time slots of lower demand.

We will first introduce some notation necessary for the discussion of the initial prices and the price update policy.

P(T): The price of a reserved bandwidth unit in time slot T.

Pmin, Pmax: The minimum price and maximum price a bandwidth unit can have.

C: The total capacity (measured in number of bandwidth units within each time slot).

D(T): The demand for time slot T, measured as a percentage of the total capacity C.

D_{thres}: A threshold demand under which P(T) is constant and equal to Pmin.

PD(T): The past average demand for time slot T.

Initial prices of forwards contracts

The initial prices for reserved bandwidth (the price of the forwards contracts) are set based on the statistical data describing past demand. For low demand – up to D_{thres} – the price of a bandwidth unit is set to Pmin. For previous demand higher than D_{thres}, the initial price is a linear function of the demand PD(T). The price may not exceed Pmax, i.e., Pmax corresponds to demand equal to the capacity C. Therefore,

$P(T) = Pmin$ for $PD(T) \leq D_{\text{thres}}$, and

$P(T) = Pmin + (Pmax-Pmin)*PD(T)/C$ otherwise

Similarly, the price for best-effort bandwidth units is initially set to Pmin/2 for past demands below D_{thres}, and to Pmin/2(1+PD(T)/C) for past demands above D_{thres}.

Price adjustment of forwards contracts

Prices are updated to reflect the knowledge about the current (real) demand D(T). For reserved bandwidth, as long as the current demand for a slot T remains below the average past demand PD(T) or below D_{thres}, P(T) does not change. If the real demand D(T) exceeds the projected past demand PD(T) (and is of course above D_{thres}), then P(T) is updated to a new value:

new $P(T) = Pmin + (Pmax-Pmin)*D(T)/C$

The price for best-effort bandwidth is not updated, since there is no way to know in advance the demand for it.

6.4 Advantages of the proposed system

The proposed system provides several advantages compared to traditional resource allocation mechanisms providing QoS guarantees. First, it provides differentiated, user-defined QoS guarantees. Traditional systems offer classes of QoS guarantees, and users are forced to chose the one closer to their requirements. In contrast, in the proposed system, the users buy the guarantees *they* want. The system does not impose any restrictions on what these guarantees are.

Second, it supports new kinds of QoS guarantees without hardware or software modifications. This is a direct consequence of customer ability to purchase the exact desired guarantees. As long as the new kinds of guarantees depend on the resources traded by the system, no changes in the hardware or software are needed to support them. It is up to the client application to purchase the appropriate resources according to the new demands.

Third, it can potentially provide better utilization of resources compared to systems with admission control. This is primarily because of two reasons. First, in most cases, fitting the user requirements to one of the offered classes of QoS guarantees offered by the system results in overestimating the required resources. Second, traditional admission control algorithms can only take few traffic parameters in consideration (e.g., peak and mean rate, delay bounds, etc.). It would otherwise become computationally infeasible to admit or reject a call in real time. Furthermore, the proposed system can continuously adapt to changes in user demands. On the other hand, centralized decision-making and scheduling of resources can take advantage of global knowledge to achieve better utilization. This is a valid concern, but its cost has to be considered. For example, one would have to change both the software (admission control algorithms) and the hardware (scheduler) to achieve the desired effects every time new knowledge can be exploited.

Fourth, users see reduced cost. Users base the purchase of resources on knowledge of their traffic characteristics and QoS requirements. Even if they do not know their exact characteristics, they can be conservative in their estimates, purchase more resources than they will actually need and then resell resources they are not going to use as soon as they find out they are not going to use them. The ability to resell unneeded resources introduces user feedback missing from today’s systems. Also, users with loose QoS requirements are able to buy cheap guarantees in the form of options and exercise their right to use guaranteed (expensive) service, only if they are in danger of not being able to achieve their QoS requirements by purchasing best-effort (cheap) service.

Fifth, time-consuming and often inefficient admission control is replaced by user decisions based on their individual QoS requirements.

Finally, new resource management algorithms can be deployed dynamically, by altering the pricing algorithms encapsulating the strategy of the service provider. In traditional systems this would potentially require new hardware support and admission algorithms.

6.5 Discussion

The proposed mechanism can be extended in many interesting directions. Examples include:

- Extension of the results to more complicated markets (e.g., markets with competing service providers and speculators
- Extension to a wider variety of QoS requirements and more sophisticated strategies of participants.
- Extension to different kinds of placement and execution of orders, such as “dynamic orders” ([36]), where parameters, conditions, or constraints will have to be satisfied simultaneously to permit the execution of an order.

- Reservation of resources that depend on each other, and provision end-to-end QoS guarantees. Specific issues include bundling of guarantees by brokers trading end-to-end assets in the markets; and introduction of timing and loose synchronization paradigms similar to those presented in [9], [10].
- Application of the mechanisms to different application domains (the mechanisms apply to a wide range of domains where agents compete for access to shared resources).

7. Schedule

I plan to complete the thesis work and defend my thesis within a year from now, by August 1999. The following is a list of things to be accomplished by then:

- Finalize the secure electronic transactions payment protocol and its prototype implementation; and provide initial measurements of its efficiency.
- Finalize the banking architecture and provide a prototype implementation of the infrastructure.
- Finalize the Msockets functions and their prototype implementation
- Study a variety of attacks; analyze how the proposed mechanisms would react to them; expand the initial results on the quantification of attack power
- Finalize the proposed mechanisms on the use of financial instruments and their use to hedge against uncertainties due to loss or congestion; provide simulation results that prove their usefulness.

Acknowledgements

Apart from my advisor Yechiam Yemini, I would like to thank the following people for very useful discussions and substantial help on the issues presented in this proposal: Doron Avital, Danilo Florissi, Gur Huberman, Jakka Sairamesh, and Hening Schulzrinne. I would also like to thank Maria Papadopouli, Ioannis Stamos, and Gong Su for their comments on the draft of this thesis proposal.

Bibliography

[1] Bellare, M., J. Garray, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner "*iKP – A family of secure electronic payment protocols*," First USENIX Workshop on Electronic Commerce, pp. 89-106, New York, July 1995.

[2] al-Binali, S. "*Competitive Analysis of Risk-Taking and Valuation of Financial Derivatives*" Ph.D. Thesis, Department of Computer Science, Columbia University, New York, NY, 1998.

[3] Bodie, Z., A. Kane, and A. Marcus *"Investments,"* third edition, 1996, IRWIN series in Finance.

[4] Chaum, D., A. Fiat, and M. Naor, "*Untraceable Electronic Cash*," Advances in Cryptology—EUROCRYPT '92 Proceedings, Springer-Verlag, 1990, pp. 319-327.

[5] Cheswick, W., and S. Bellovin "*Firewalls and Internet Security: Repelling the Wily Hacker*," Addison-Wesley, Reading, Mass. 1994.

[6] Clearwater, S., editor. *"Market-based Control of Distributed Systems,"* World Scientific Press, 1996

[7] CyberCash, http://www.cybercash.com/

[8] Digicash, *http://www.digicash.com/*

[9] Florissi, D. "*Isochronets: a high-speed network switching architecture*," Ph.D. Thesis, Department of Computer Science, Columbia University, 1995.

[10] Florissi, D., and Y. Yemini "*Protocols for loosely synchronous networks*," Proceeding of the 4th International IFIP Workshop on Protocols for High Speed Networks, August 1994.

[11] Gedra, T. and P. Varaiya "*Markets and Pricing for Interruptible Electric Power*," IEEE Transactions on Power Systems, Vol. 8, No. 1, February 1993.

[12] Hull, J. C. *"Options, Futures, and Other Derivatives,"* third edition, Prentice Hall.

[13] Jarecki, S., and A. Odlyzko "*An Efficient Micropayment System Based on Probabilistic Polling*," in Proc. of Financial Cryptography'97, R. Hirschfeld, ed., Lecture Notes in Computer Science, Springer, 1997.

[14] Ilgun, K, R. Kemmerer, and P. Porras "*State Transition Analysis: A Rule-based Intrusion Detection Approach*," IEEE Transactions on Software Engineering, 21(3), pp. 181-199, March 1995.

[15] Kaufman, C., R. Perlman, and M. Speciner "*Network Security – Private Communication in a Public World*"Prentice Hall series in computer networking and distributed systems, 1995.

[16] Kohl, J. and C. Neuman "*The Kerberos Network Authentication Service (V5),*" RFC 1510, Sept. 1993.

[17] Kurose, J., M. Schwartz, and Y. Yemini *"A Microeconomic Approach to Optimization of Channel Access Policies in Multiaccess Networks,"* Proc. of the 5th International Conference on Distributed Computer Systems, Denver, Colorado, 1995.

[18] Lee, W. and Stolfo, S. "*Mining Audit Data to Build Intrusion Detection Models*,"

[19] Lunt, T, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey "*A Real-time Intrusion Detection Expert System (IDES) – final technical report*," Technical Report, Computer Science Laboratory, SRI International, Menlo Park, California, February 1992.

[20] MacKie-Mason, J., and H. Varian *"Economic FAQs About the Internet,"* in The Journal of Economic Perspectives, vol. 8, no. 3, pp. 75-96, 1994. Reprinted (with revisions) in Internet Economics, J. Bailey and L. McKnight, eds. Cambridge, MIT Press, 1996.

[21] MacKie-Mason, J., and H. Varian *"Pricing the Internet,"* in B. Kahin and J. Keller, editors, *Public Access to the Internet*, ACM, Boston, Massachusetts, May 1993.

[22] Manasse, M. "*The Millicent protocols for electronic commerce*," Proceedings of the first USENIX Workshop on Electronic Commerce, New York, July 1995

[23] Mondex, http://www.mondex.com/

[24] Needham, R., and M. Schroeder "*Using Encryption for Authentication in Large Networks of Computers*," Communications of ACM, Vol. 21, Dec. 1978, pp. 993-999.

[25] Needham, R., and M. Schroeder "*Authentication Revisited*," Operating Systems Review, Vol. 21 # 1, Jan. 1987.

[26] Netbill, http://www.netbill.com/netbill/protocol.html

[27] Okamoto, T. and K. Ohta "*Universal Electronic Cash*," Advances in Cryptography, CRYPTO'91 Proceedings, Springer-Verlag, 1992, pp. 324-337.

[28] Pedersen, T. "*Electronic Payment of Small Amounts*," Cambridge Workshop on Security Protocols, 1996.

[29] Sairamesh, J., D. Ferguson, and Y. Yemini *"An Approach to Pricing, Optimal Allocation and Quality of Service Provisioning in High-speed Packet Networks,"* in Proc. of the Conference on Computer Communications, Boston, Massachusetts, April 1995.

[30] Sairamesh, J., and J. Kephart "*Dynamics of Price and Quality Differentiation in Information and Computational Markets*," Proceedings of First International Conference on Information and Computation Economies, ICE98, Charleston, SC, Oct. 1998.

[31] Schneier, B. "*Applied Cryptography*," second edition, John Wiley & Sons, pp. 139-147.

[32] "*Secure Electronic Transactions: Credit Card Payment on the Web in Theory and Practice*," IBM International Technical Support Organization, June 1997.

[33] Spafford, E. "*The Internet Worm Incident*," Technical Report CSD-TR-933, Department of Computer Sciences, Purdue University, Sept. 19, 1991.

[34] Stamos, I., P. Allen "*Interactive Sensor Planning,*" Proc. Computer Vision and Pattern Recognition, Santa Barbara, CA, June 1998.

[35] Walsh, W., M. Wellman, P. Wurman, and J. MacKie-Mason *"Some Economics of Market-Based Distributed Scheduling,"* In Proc. of the 8th International Conference on Distributed Computing Systems (ICDCS-98), Amsterdam, the Netherlands, May 1998.

[36] Wellman, S. "*Technology takes to securities trading*," IEEE Spectrum, Feb. 1997.

[37] Yemini, Y. *"Selfish Optimization in Computer Networks,"* Proc. of the 20th IEEE Conference on Decision and Control, pp. 281-285, San Diego, CA., Dec. 1981.

[38] Zhang, L., S. Deering, D. Estrin, S. Shenker, and D. Zappala *"RSVP: A New Resource ReSerVation Protocol,"* IEEE Network magazine, 7(5):8-18, September 1993.