Mitigating Distributed Denial of Service Attacks with Dynamic Resource Pricing

David Mankins, Rajesh Krishnan, Ceilyn Boyd, John Zao, Michael Frentz
BBN Technologies
dm@bbn.com, krash@bbn.com, boydz@mindspring.com, jzao@bbn.com, mfrentz@bbn.com

Abstract

Distributed Denial of Service (DDoS) attacks exploit the acute imbalance between client and server workloads to cause devastation to the service providers. We propose a distributed gateway architecture and a payment protocol that imposes dynamically changing prices on both network, server, and information resources in order to push some cost of initiating service requests — in terms of monetary payments and/or computational burdens — back onto the requesting clients. By employing different price and purchase functions, the architecture can provide service quality differentiation and furthermore, select good client behavior and discriminate against adversarial behavior. If confirmed by additional experiments, judicious partitioning of resources using different pricing functions can improve overall service survivability.[1]

1. Introduction

Denial of Service continues to be a pervasive problem for Internet services, as evidenced by the recent denial of service at CERT **[BBC01]**. Such attacks require each attacking machine to perform only a small amount of work, relying on the cumulative efforts many machines to overload the victim machine. Attacks range from simple ICMP ping requests to sophisticated attacks that are difficult to distinguish from a sudden spike in legitimate use (a.k.a. "flash crowd" [**Niv73**] or "the Slashdot Effect").

In a DDoS attack, the perpetrator(s) may spend weeks or months subverting hundreds or thousands of machines by exploiting well-known security flaws. Once the machines are subverted, the perpetrator installs tools to execute an attack. On command, the prepared machines (known as "zombies") collectively target a specified victim with a packet storm consisting of repeated datagram packet requests. These packets may take many forms, such as an ICMP ping request, a UDP packet (such as a DNS request), or TCP SYN floods. The packets may also have forged return addresses, allowing one machine to generate requests that appear to be coming from hundreds of machines, and making the sources of the attack difficult to trace. This problem may intensify since newer operating systems that are being introduced may make it easier to spoof IP addresses [**Gib01**]. Halting these attacks is difficult and typically relies on filtering at the router (or just waiting out the "packet storm").

Packet-flood attacks, such as these, are the most widely diagnosed today **[CAIDA01].** As measures to counteract them are put in place, future attacks will most likely include resource depletion attacks that are more difficult to distinguish from legitimate requests. The effectiveness of an attack will be driven by the resource most tightly constrained relative to the attack parameters. The attack could focus on a single server within an enclave (or on the pipes themselves).

In this paper, we explore the use of dynamic resource pricing strategies to mitigate various DDoS attacks and to improve service survivability. We propose an experimental architecture and protocol for a Market-based Service Quality Differentiation (MbSQD) system. We have prototyped the MbSQD system using the ns-2 simulator [**NS2**], and have experimented with various price-based controls for DDoS mitigation. These include both proof-of-work based approaches that exact a price in terms of a computational burden or monetary-like micropayments. The MbSQD system is designed to work as an overlay on existing network security infrastructure, but similar concepts could also be applied at the lower networking layers (e.g. in conjunction with Diffserv/ RSVP).

1.1 Existing DDoS mitigation strategies

Various methods have been used to mitigate the effects of DDoS attacks – each has its drawbacks and advantages. Some of these methods are:

- Detecting and eliminating subverted nodes on local subnets

[1] This research is supported by DARPA contract F30602-00-C-0088

- Choking off the flow of attack packets
- Using resource mirrors to amplify server capacity
- Reducing the amount of resources consumed
- Monitoring traffic volume
- Tracing attack packets though the Internet

Detecting subverted nodes has become increasingly difficult as many of the newer attacks encrypt commands to hide them from security scanning processes.

Stopping the flow of attack packets at a choke point relies on ingress or egress filtering. The effectiveness of these techniques is dependent on good practices by the ISPs and may require active maintenance by administrators at the leaf routers of a customer site. Additionally, these methods may discard legitimate packets during asymmetric or changing routing conditions.

Establishing resource mirrors, so that the attacks affect only a fraction of the users of the service, can lessen the impact of attacks (e.g. Akamai's method of caching content in "edge proxies" scattered around the edges of the net), however this solution is relatively expensive and may be reasonable only for larger content providers.

There are various methods that attempt to reduce the amount of resources consumed during a DDoS attack. NetBSD will accelerate its timeouts of partially opened TCP connections at times of contention. Linux implementations save no state when receiving a TCP SYN – instead, the initial sequence number is derived from information that will be available in the return packet from the connecting machine (SYN Cookies). The effectiveness and availability of these techniques is highly implementation dependent.

Monitoring traffic volume to look for suspicious traffic destined for a particular destination can be limited by traffic flow confidentiality techniques.

Attacking DDoS Packets may be traced backwards to their source by augmenting the capabilities of routers. Responsible ISPs or system administrators can be contacted to shut the attacks off at their sources (see for example, [**Sno01**]). Such mechanisms work after-the-fact; however, they may serve as deterrents.

2. Technical Approach

2.1 General Observations about DDoS Attacks

Denial-of-service attacks create shortages of a resource such as bandwidth or computing cycles through the creation of an artificial demand. They work because the "cost" of the transaction falls overwhelmingly on the server. Sophisticated DDoS attacks also can be virtually indistinguishable from genuine overload (at least at the time of the attack) due to the limitations of the information available and the kinds of analysis possible in real-time. A mechanism is necessary to transfer a corresponding burden to the requesting client and to control the damage that any single client can cause. This characterization suggests that an economics-based approach to establish a "marketplace" for the services may provide a fairer allocation of resources, provided that the currency's availability can be effectively allocated by the service provider to its customers (and kept away from its detractors).

To make services more robust against a DDoS attack, we propose the following combination of strategies:

1. Increase the barrier to entry by using a pricing-based scheme in which the price of entry varies with the load level. This will throttle the machines used in the attack, thereby forcing the attacker to employ (or subvert) a larger number of machines.
2. Use a differentiated model. Provide prioritized access to classes of users; though a DDoS attack will raise the price so high that lower priority classes get locked out, higher priority clients can still access the service. Allocating a priority mechanism to desirable clients is key.
3. Use a dynamic, differential pricing mechanism to penalize clients that are responsible for a load on the server. This typically requires flow monitoring and isolation capabilities in line with those of Diffserv [**Ful00**][**Arq**] [**Bla98**].

None of these strategies are sufficient in isolation — situations can be defined where each may contribute to increasing the availability of the network services.

2.2 Types of Micro-payments

Micro-payments can provide a useful side benefit by providing a uniform means of resource accounting, pricing, and arbitration. Micro-payment mechanisms must not impose an undue performance penalty – in the absence of an attack, the performance should be nearly comparable to a system that does not use the payment mechanisms. There is prior work on how pricing can be used to influence consumer behavior, how to integrate pricing mechanisms with OS and network resource management mechanisms. In this paper, we instead focus on how pricing strategies can be used to mitigate DDoS, and improve overall service survivability.

2.2.1 Fungible vs. Non-fungible Micro-payments

There have been a number of digital payment and micro-payment schemes proposed to support digital exchanges [**Riv97**]. These have been primarily proposed to support digital commerce, but some researchers have also looked at the use of payment schemes as a means of mitigating denials of service.

Fungible (or transferable) digital payment schemes range from anonymous cash schemes such as David Chaum's Digicash [**Digi**] [**Cha83**], to electronic checks and payments [**Pay**], postage [**Hash**], to bartering services (as in Mojo Nation [**Mojo**]).

Cash-like schemes do not require on-line verification – the server can validate the coin by examining it. They therefore have low latency with respect to coordination with external servers, however the validation process typically requires significant computation or memory usage overhead for the server itself. As a result, high integrity cash-like payment schemes may not be compatible with fielded servers.

Many of the alternative fungible payment schemes are analogous to a check or credit card transaction and require some type of on-line verification of payment – a server must connect online with a bank and verify the creditworthiness of the requester. On-line verification is susceptible to high latency and provides an alternative critical path target for DoS.

Scrip-based systems are an attempt to reduce the latency of verification by making the verification a purely local operation on the server. These systems, such as Compaq's Millicent [**Milli**], are intended for ultra-micro-payments (on the order of thousandths of a penny). In Millicent, a server issues (or mints) its own scrip to be used by clients to pay for services. Since the server issues the scrip, it can verify it with very low latency (possibly requiring as little as a table lookup). Clients obtain a quantity of scrip from network *scrip brokers* using one of the high-overhead bulk-payment schemes geared for larger expenditures. Millicent allows a server to give a client *change* (which the client may later redeem with their broker).

Another way of escaping the need of on-line verification is to extract a payment in the form of "work" or computation [**Dwo92**][**Jue99**][**Jak99**][**Hash]**. The server sets a computational task to the client that must be solved before server resources are expended on the client's task. To be useful, the task must be computationally hard to solve, but a solution must be simple to verify e.g., factoring a large number — the size of the number is determined by the prices of the service, which in turn reflects the load (demand) on the server.

2.2.2 Convertible vs. Non-convertible Currencies

A *non-convertible* currency scheme has a limited scope where it can be used and cannot be exchanged for other types of currency. A *convertible* currency on the other hand can be exchanged for other types of currency.

The former is useful to permit priority access to specific resources for a particular subset of known potential users (e.g. a military squadron). The latter has advantages in situations requiring high priority access for a dynamically changing subset of potential users drawn from a general population.

2.3 Dynamic Resource Pricing as Discriminants

As the logical next step, we implement a dynamic pricing strategy that can favor good user behavior and discriminate against aggressive adversarial behavior.

In our model, we have a time-varying price function for each service. The price function relates the price of the service to supply, demand and other factors.

Each user has a utility function that determines how much they are willing to pay for a unit of a given service as well as how many units they will consume at a given price at any given time. The cumulative effect of the utility functions drives the overall demand for the service. Furthermore, the spending behavior can be monitored in a distributed fashion for anomalies.

Selection of one particular user behavior over another occurs due to interplay of the price and utility curves. While the idea of using pricing to mold user behavior is known, our approach extends this idea to discriminate against adversarial behavior.

Consider the following scenario of a web server. A number of different kinds of attacks can be launched against a web server. These include exploiting OS and protocol stack vulnerabilities such as SYN floods and buffer overflows, connection depletion attacks using idle connections, depleting server resources using requests that are expensive to process, inundating the request queue with bogus connections, and depleting network bandwidth by requesting large volumes of different pricing strategies are required to protect against each of these attacks,.

A robust pricing function may monitor numerous user indicators. Even if individual users remain anonymous, control can be exercised at the granularity at which the flows can be isolated and monitored. For example, pricing controls can still be exerted at the level of the originating ISP. Resources can be partitioned between anonymous users and long-term subscribers. ISPs can protect themselves by using similar price controls and other monitoring within their administrative domains. Adoption of egress filtering and IP trace back can further aid in enforcing such controls.

Pricing functions must be robust against any potential new attacks enabled due to the pricing strategy itself. For example, care must be taken that an adversary is not able to populate the system with fake requests when the price is low and increase the price for legitimate users. This means that a pricing function that is robust against connection depletion attacks must necessarily limit the

connection duration and require that each connection be refreshed periodically to protect against zombies.

2.4 Price-based Service Quality Differentiation and Survivability

We have discussed how different pricing functions can be used to select different kinds of user behavior, thereby protecting against some classes of attacks. A single price function is unlikely to provide sufficient service quality differentiation necessary to satisfy a wide range of user requirements. Furthermore, a single pricing function is unlikely to protect the system against all forms of attack.

Therefore we propose to partition and isolate the available resources among various service classes. For example, in the case of a network router, weighted fair queuing can be used to partition resources. Resources can also be isolated using VPNs. Server resources can similarly be partitioned and isolated among service classes by using OS prioritized scheduling techniques and by virtual OS techniques respectively.

We hypothesize that the survivability of the system can be further enhanced by associating different partitions with different discriminants (pricing functions) that are robust against different classes of attacks. With this approach, a successful resource depletion attack will not only require more resources, but also the simultaneous launch of different forms of attack for each service class.

Possible extensions of this strategy include dynamic policy iteration that progressively improves robustness against a larger class of attacks or a randomized policy iteration that makes it harder for the adversary to guess the pricing function and determine efficient attack strategies.

3. System Architecture

Figure 3-1 illustrates the operational architecture for the MbSQD system. The MbSQD system employs a distributed architecture with three distinct features:

1. *Deployment of resource brokers at network boundaries:* MbSQD will use stateful packet filters and/or application proxies to control resource utilization at the logical boundaries of user subnets (on either the provider's or the client's sites). This architecture has the following advantages:
   a) The operation of client and server applications will not be affected by the deployment of the traffic control system; in fact, both clients and servers may not be aware of its presence except due to apparent changes in network throughput and device performance. No modification of end-node protocols and applications is necessary.
   b) The architecture may be used to control the utilization of both network and information resources including network throughput, server capacity, information access and device usage. The brokers may be installed at the border gateways of autonomous systems if they are intended to be used for inter-domain traffic control, or they can be placed at the "choke points" of server access if they are used to control information access and/or device usage.
   c) Price-based resource management can be made *mandatory* in order to obtain the highest priority access privileges.
2. *Employment of client-side defined price and purchase decision functions:* MbSQD achieves rapid control of resource utilization by relying on the interactions between the dynamic pricing of resources and the autonomic purchase decisions made by individual clients. By employing different pricing functions, MbSQD can favor the clients that exhibit desired behaviors or use certain forms of purchase decision functions. This behavioral discrimination is a unique feature of the dynamic pricing scheme.
3. *Operation with TCP-integrated payment protocols:* MbSQD uses a three-message handshake protocol to initiate service request and conduct payment transaction; it also uses two-message handshakes to pay for continuous resource use. The initial payment protocol can be readily integrated with the TCP connection establishment, and the renewal handshakes can be "piggy-backed" onto TCP data segments as options. The protocols are also designed to support different forms of payments including scrip and proof-of-work. A micro-payment infrastructure will be needed in order to use scrip.

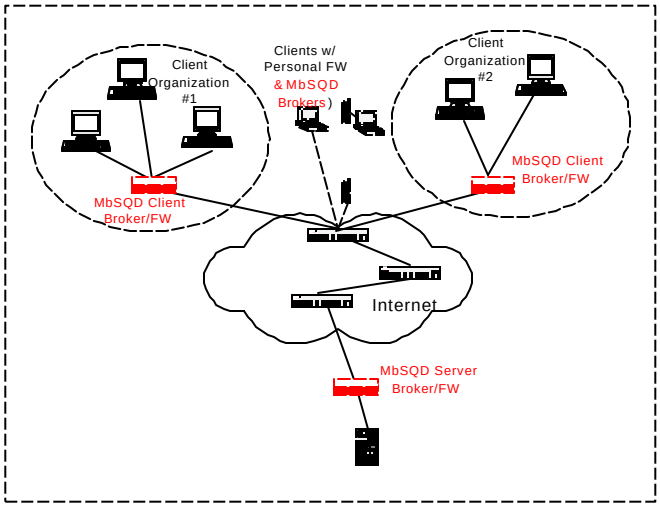

Figure 3-1. MbSQD Operational Architecture

In the remaining parts of this section, we will examine the three essential components of MbSQD: the resource brokers that are installed at the boundary gateways, the business logic that implements the price and the purchase decision functions, and the payment protocol that enables the business transactions.

3.1 Resource Brokers

The MbSQD resource brokers are to be installed in gateways at the boundary of Internet sub-networks, where they can function as application proxies or packet filters. The brokers determine whether datagrams going to and from certain *IP addresses* using specific *transport protocols* and *port numbers* should be passed or discarded. Functionally, the resource brokers may be the proxies of application clients or servers. The server and client brokers enable passage of datagrams based resource prices and budgetary considerations. Operationally, each broker consists of two sets of components that either operate on *data* or *control flows*.

3.1.1 Server vs. Client Brokers

The client brokers function as proxies of the client applications that run on the end hosts. The client brokers submit the requests for services — specified in terms of server IP addresses, transport protocols and port numbers — to the server broker on behalf of the client applications, make the purchase decision when the server brokers reveal the current prices of the services and conduct the transaction in order to establish passages for the traffic.

The server brokers, on the other hand, function as proxies of the server applications that provide specific services. The brokers determine the dynamic prices of the services based on several traffic parameters that are monitored continuously. They also work with the client brokers to conduct the payment transactions and control the client-server traffic flows.

3.1.2 Control vs. Data Flows

Each client or server broker consists of four components: *traffic classifier, traffic monitor, business logic* and *business executive* as shown in Figure 3-2.

- The *traffic classifier* redirects IP datagrams according to the nature of their payloads so as to separate control and data flows. It also determines whether to pass or discard datagrams in the data flows based on the outcome of payment transactions.
- The *traffic monitor* provides the values of traffic parameters that are used to establish the current prices of specific services and/or serve as the indicators of anomalous traffic behaviors.
- The *business logic* is the decision making component that computes the service prices in a server broker and make the purchase decisions in a client broker.
- The *business executive* is the module that conducts the payment transactions using micro-payment protocols and controls the traffic classifier.

3.2 Business Logic

The business logic is a collection of rules and associated parameters that are used to control the traffic flow related to a service. It is designed to protect the resources such as network bandwidth or server capacity that a service deems important. Service providers must decide how to price and sell the services that they offer and then configure the business logics for each service. In MbSQD, services are distinguished by their server IP addresses, transport protocols and port numbers.

A client gains access to a service running on a specific server by purchasing a *subscription* to that service. Service providers are free to decide how to price the service — in packets, seconds or by connections. By carefully choosing the pricing algorithm and the units in which the subscription is sold, a service provider can manage the utilization of its service so that "good" client behavior is rewarded and "bad" client behavior is

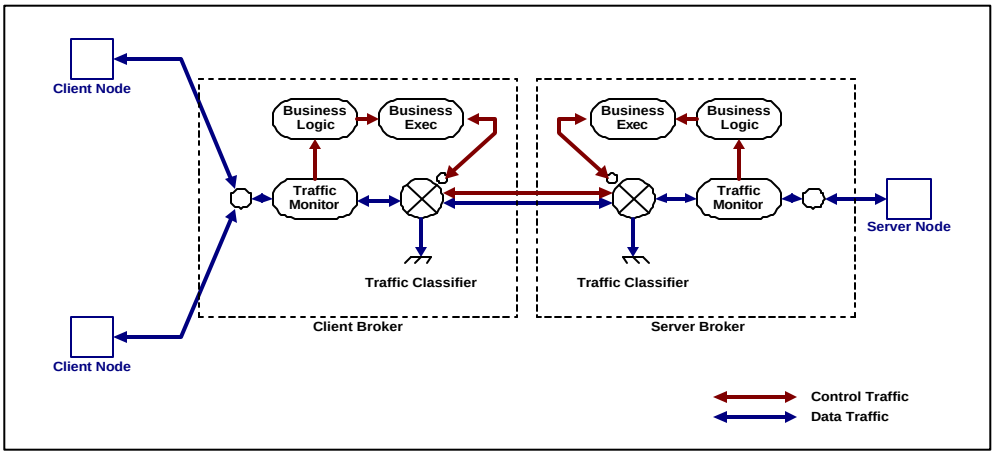

Figure 3-2. Functional Architecture of Resource Brokers

penalized based on some a priori definition of "good" and "bad". Note that the design of the business logic is independent of that of the payment mechanism. The current design of MbSQD can use scrip and proof-of-work computation as payment tokens.

The rest of this section discusses briefly the *subscription types* — i.e. how the resources are quantified and sold — as well as the *price* and *purchase decision functions.*

3.2.1 Subscription Types

Currently, four general subscription types are implemented in MbSQD.

a) *Subscriptions in Packets:* subscriptions are offered to customers on a per packet basis. The service provider defines a maximum number of packets the client can send or receive; once the quota has been met, the subscription expires and the client must pay for additional service.

b) *Subscriptions in Seconds*: subscriptions are sold in seconds of connection time. When the connection duration has elapsed, the subscription expires and customers must purchase a new subscription. A time-based subscription may be used in conjunction with another subscription type to create a hybrid subscription type. For instance the subscriptions may be sold in terms of number of packets, but a client must send or receive the packets within a certain period of time. Such a hybrid type may be useful discourage clients from "squatting" on a connection.

c) *Subscriptions in Connections:* a client pays for a connection that lasts for an indeterminate duration. This subscription type may be combined with a time-based subscription to simulate leasing of a resource.

d) *Subscriptions in Bytes*: a client may also purchase a subscription based on the number of bytes sent to or from a server.

3.2.2 Pricing Functions

The pricing strategy is the mechanism that the server broker uses to control resource consumption. Whenever a new subscription request for a service arrives at the broker, it invokes the business logic of the service to calculate a price for the new subscription based on the current values of the *market observable* defined for the service.

In our experiments, we tested the following four different forms of pricing functions:

Constant Function ($p = k$): the price p of the resource is set to constant k regardless of its level of consumption.

Linear Function ($p = kc$): the resource price p is proportional to the value of a chosen market observable c such as the number of current connections.

Asymptotic Function ($p = kB/(B-c)$): this function raises the resource price p to infinity as the market observable c approaches its limiting value B; such a pricing strategy is useful in safeguarding a resource with a hard limit in capacity.

Exponential Function ($p = k_1 e^{k_2 c}$): this function produces the fastest increase of resource price with respect to the increasing value of the market observable c; such a pricing strategy is useful in controlling consumption of a critical resource.

Each of these pricing functions produces a floating point number that can translated into appropriate proof-of-work computation or scrip values.

3.2.3 Purchase Decision Functions

At the client brokers, interacting with the resource prices are the *purchase decision functions*, which determine whether to purchase the subscriptions by making required payments. The decision functions can employ sophisticated strategies based on the market observables and other parameters supplied by the clients. The simplest decision function might only specify a price ceiling for each client.

3.3 Micro-payment Protocols

MbSQD employs a three-message handshake to perform the payment transactions. The handshakes provide a framework protocol that can be used to support different forms of micro-payment including scrip and computational proof-of-work. It can also be made compatible with various micro-payment infrastructures.

Figure 3-3 shows the message sequence of the payment protocol. It uses three messages (solid lines) to submit the service/subscription request and exchange the payment

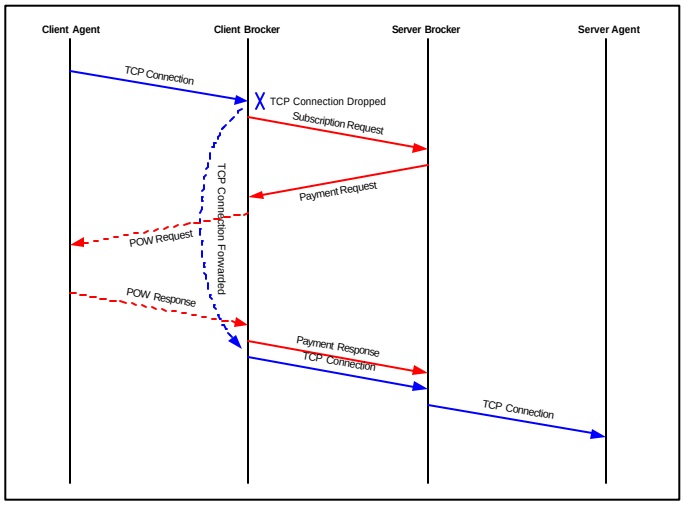

Figure 3-3. Message sequence of payment protocol

request and response. These messages can be readily integrated with the TCP connection-establishing messages as options in the respective SYN messages. The payment response message may also be integrated as an option into the ACK message of the initiator.

In order to support proof-of-work, two more messages (dotted lines) are added between the client broker and the clients. These messages complete a handshake that will be completed if and only if the proof-of-work computation is carried out successfully.

4. Simulation Experiments

We investigated the behaviors of MbSQD broker architecture and traffic management mechanisms by conducting a series of simulation experiments using public-domain discrete-event network simulator, ns-2. The experiment configuration is shown in Figure 4-1.

In the experiments, a fixed set of legitimate clients was programmed to request the service offered by a single server. Their requests were mingled with much larger number of requests initiated by the rogue clients that were subverted to instigate DDoS attacks. Client and server brokers, deployed at the boundaries of the sub-networks that contain the clients and the servers, relayed the service requests of both "good" and "bad" clients. Both client and server brokers could operate in an active or an inactive mode. In the inactive mode, the brokers behaved like ordinary firewalls or routers. When activated, client and server brokers could control the traffic flowing through them by matching the IP datagrams with the connections established between the brokers. The datagrams were passed if they could be matched with one or more of the established connections and discarded otherwise. In each run, we observed the progress of two DDoS attacks – one with and the other without the use of MbSQD. Control parameters, such numbers of rogue clients and traffic characteristics, were changed between experimental runs.

The goal of the experiments was to investigate the *effectiveness* of MbSQD architecture in mitigating the DDoS attacks. The *degree of effectiveness* was inferred from the following two sets of observations:

1. A comparison between the number of subverted clients required to launch two similar DDoS attacks that cause compatible levels of performance degradation with and without the activation of MbSQD brokers;
2. A comparison of the residual level of services available to the fixed set of legitimate clients in the two similar DDoS attacks with and without MbSQD brokers.

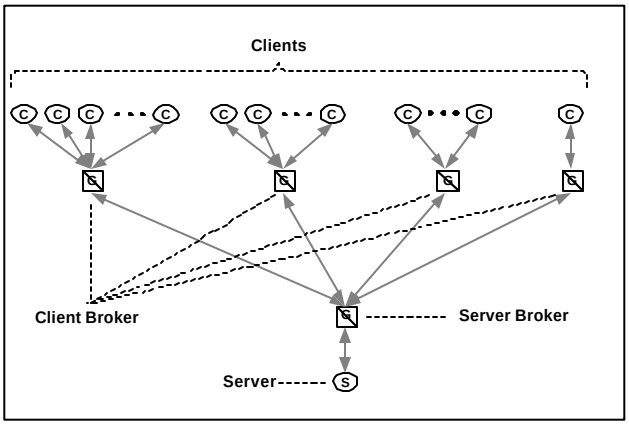

Figure 4-1. Configuration of simulation experiments

A secondary goal was to investigate the *usefulness* of price or other market parameters employed in MbSQD as *indicators* of presence or absence of possible attacks and the severity level of the attacks.

4.1 Metrics

In order to establish a quantitative description of DDoS attacks, and an economic model for client-server traffic, we introduced three sets of parameters into MbSQD, known as *attack quantifiers*, *market observables* and *market controllables*.

4.1.1 Attack Quantifiers

The effect of a DDoS attack may be quantified by two sets of measurements:

Service Quality: as its name indicated, this measurement reflects the quality of a specific *service* received by a specific *client* or *group of clients*. Assuming web browsing is the service of interest and the DDoS attack aims at exhausting the number of active HTTP sessions maintained by the server, a suitable measurement for the service quality would be a pair of values consisting of the average latency for establishing a new HTTP session and the average data rate sustainable by a legitimate client under no-load condition.

Attack Duration: currently, a DDoS attack may last as long as the attackers intended. By introducing a payment scheme and providing each client with limited amount of credits, MbSQD established an inherent limit to the amount of service any one client may obtain before spending all its credits, and thus created a natural end to an attack. In the experiments, we monitored the progress of an attack by collecting measurements periodically throughout its course.

4.1.2 Market Controllables

In MbSQD, the trading of resources is controlled by the server broker through setting of three control parameters.

Connection Granularity: the connections established by the server brokers may cover a group of clients and a range of protocols. Selectors such as source/destination IP addresses, transport protocol identifiers and TCP/UDP port numbers serve to specify the granularity of the connections.

Connection Duration: the connections established between client and server brokers could only last for a finite duration. This parameter specifies the duration of the connections established at the current moment. Note that the duration may be specified in real time or data size (either in number of bytes or packets).

Connection Price: this parameter refers to the time-varying price for a connection that is set by the pricing function maintained in the server broker.

4.1.3 Market Observables

The price of a resource was computed based on the current values of a selected set of parameters that reflect resource consumption. The choices of these parameters are determined largely by the nature of the resources and the strategy employed to manage those resources. Following three parameters were used in our experiments to determine the price of HTTP connections.

Connection Request Count, which records the total number of HTTP requests received by the server broker within a measurement period.

Connection Establishment Count, which records the total number of HTTP connections established by the brokers within a measurement period.

Data Throughput, which records the total number of data bytes passed to the server within the past second; the measurement is also computed as a percentage of the maximum data rate sustainable by the server and the data link between it and the server broker.

4.2 Results

We performed three sets of experiments that were designed to study the behaviors of MbSQD system in response to three different kinds of DDoS attacks:

1. *TCP-SYN Attacks:* in these experiments, the rogue clients flood the server with SYN packets with forged source IP addresses in order to overwhelm the server with half-opened TCP connections;
2. *Server Flooding Attacks:* in these experiments, the rogue clients flood the server with frequent and long TCP connections uploading large amount of data to the server; this set of experiments were also designed to examine the effects of using computational proof-of-work as a method of payments offered by the clients;
3. *Server Draining Attack:* in these experiments, the rogue clients initiate frequent TCP connections downloading large amount of data from the server (e.g. an HTTP server). This set of experiments also examines the effects of using fungible payments as a means of payments.

Experiments of type (1) and (2) were run with 25 legitimate clients requesting TCP connections of random exponentially distributed durations (average 0.5 seconds) separated by random exponentially distributed intervals (average 0.1 seconds). Each of the legitimate clients would establish a connection with the server and upload a relatively small amount of data. Experiments were run with from 0 to 4000 rogue clients. The rogue clients were identical to the legitimate clients except that they were more aggressive: their average interval was shorter (0.1 seconds) and the average duration of their data uploads was also larger (0.7 seconds).

Experiments of type (3) were run with 128 legitimate clients opening TCP connections and sending requests requiring a response of a random, exponentially distributed size (average 17,000 bytes) at exponentially distributed intervals (average 0.1 seconds). Experiments were run with from 0 to 512 rogue clients. The rogue clients loop continuously sending requests requiring random, exponentially distributed sizes of a larger average (average 100,000 bytes).

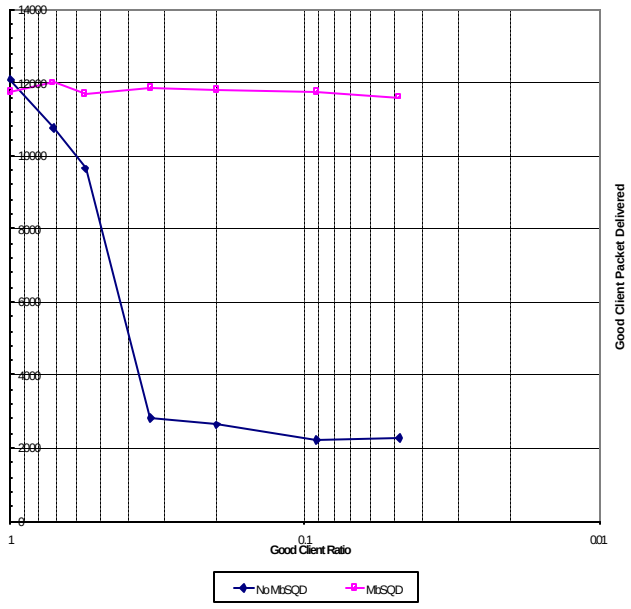

Figure 4-2. MbSQD mitigation against TCP-SYN attacks

4.2.1 Mitigation against TCP-SYN Attacks

In these experiments, the rogue clients were programmed as constant bit sources, generating SYN packets as fast as they can without completing the connection establishment. The attack did not start until 0.5 seconds into the experiment, and stopped 1.5 seconds before the end of the test. Figure 4-2 displays the result of the experiment. The graph plots two traces: the number of packets from the legitimate clients that were delivered to the server agent while MbSQD was running verses the same measurement without MbSQD. The vertical axis shows the total packet count (over the course of the entire experiment). The horizontal axis is the ratio of Good Clients to DDoS attacking agents. As the graph shows, the intensity of the attack has little impact on the throughput of the legitimate clients when MbSQD is active; on the other hand, the service is effectively denied in the absence of MbSQD.

These drastically different results have a simple explanation. By requiring a proof-of-work response from the clients before the brokers would forward packets, the impact of the SYN attacks is shifted from the server to the gateways that host the brokers. A naive SYN packet flood has no impact on the server because the server broker discards all the attacker packets without burdening the server with them.

4.2.2 Mitigation against Server Flooding

In these experiments, the rogue clients had similar but more aggressive behaviors than the legitimate clients: their average connection interval was set to be 0.01 second and their average duration was 0.7 second; in other words, the rogue clients requested connections much more frequently

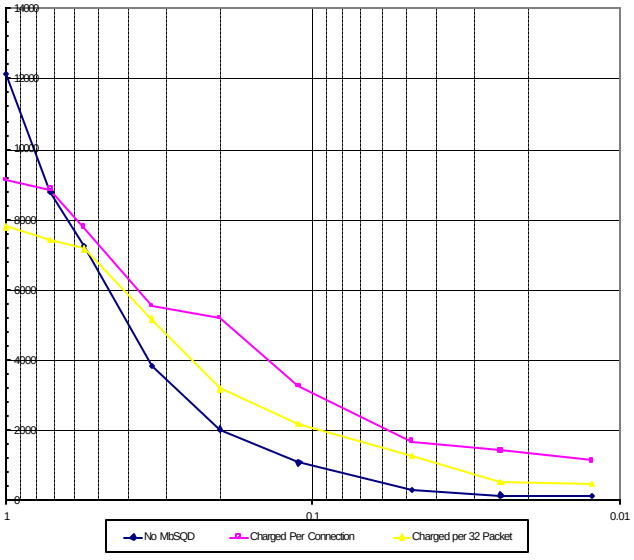

Figure 4-3. MbSQD mitigation against server flooding

than the legitimate clients, and held onto them for a little longer. The resource protected by the server broker includes the server node, the server broker, and the link between them. In these experiments, the price was linearly proportional to the number of open connections at the server broker (and at the server). The token of payment was proof-of-work computation that the clients must perform in response to the challenges from the client broker.

We ran three sets of simulations: a control case without MbSQD action, one with the server broker charging a price for the establishment of each connection and the other one with the server broker charging a price for every pass of 32 packets. As a measure of service quality, we counted the number of legitimate-client requests delivered to the server during ten seconds of simulation time. (We also verified that the total number of requests delivered to the server remained at 10000, no matter how many rogue clients were present.) Figure 4-3 displays the results of the experiment.

The graph shows that all attacks were ultimately effective even with the use of MbSQD. However, it required eight times as many rogue clients in order to achieve the same level of service degradation when the brokers were active. Since our pricing strategy throttled both legitimate and rogue clients, the effects were not quite as dramatic as those of the first set of experiments, but it remains the case that an attacker must infect two to three times as many hosts in order to achieve and sustain

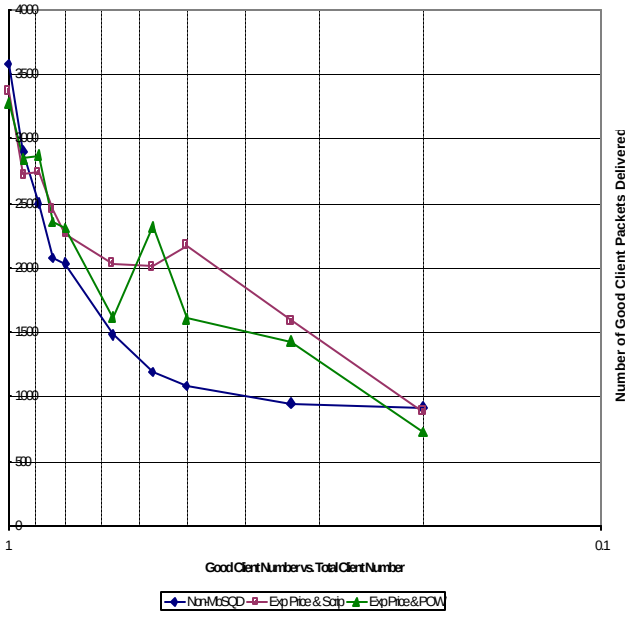

Figure 4-4. MbSQD mitigation against server draining

a significant level of service degradation.

4.2.3 Mitigation against Server Draining

In this experiment, the results of which are shown in Figure 4-4, the rogue clients are attempting to act as a drain on a server's bandwidth. They attempt to repeatedly download large files, exhausting the number of connections available for legitimate requests (the size of their download is large simply to let them sit on the connection once they have it).

In this experiment, users leased connections. The lease period was set so that, in an unloaded system, 90% of all legitimate tasks would complete requiring only one payment.

That the effectiveness of this price function diminishes under heavy load comes as no surprise – as the load increases, a constant time lease covers less data. An experiment based on amount of data transmitted (as opposed to connection duration) should prove more robust at the higher attack rates.

5. Conclusions

In this paper, we have explored the application of dynamic pricing mechanisms in mitigating DDoS attacks. We have presented the MbSQD architecture and protocol which supports both proof-of-work and monetary-like micropayment schemes. We have prototyped the MbSQD system using the ns-2 simulator. We also presented simulation results on the effectiveness of different pricing strategies for some DDoS scenarios based on a monopolistic service model.

5.1 Lessons learned

We made the following observations from the simulation experiments we have conducted:

- Pushing costs back onto clients appears to be effective for mitigating server-based DDoS attacks. Specifically, MbSQD does show promise for maintaining control of client-server traffic flows over the Internet. Traffic parameters can be implicitly exploited to maintain control even under changing conditions
- Proof-of-work methods are effective for elimination of spoofed requests or flooding via a limited number of machines. Scrip based payment methods can be effective if integrity can be maintained over the money supply.
- Different client behaviors can be discriminated by different server pricing strategies. Service Brokers can work to favor certain traffic behaviors in the range of scenarios we have studied.
- As to be expected, the choice of a pricing function will have a very strong effect on the effectiveness of MbSQD. Pricing functions can favor either the defender or the attacker and care must be taken to choose pricing functions that elicit behavior conducive to the accomplishment of the mission of the server(s) being controlled.

5.2 Suggested Directions for Future Research

The investigation presented in this paper is preliminary in nature. We suggest the following two possible extensions of the current system.

5.2.1 Dynamic Subscription Parameters

Currently, subscription parameters types and limiting values are fixed throughout the process. However, a service provider might define a business logic in which lengthy subscriptions are offered when the service is not busy, but allowed length may shrink in response to service load. Alternately, a service might offer free service until its resources are depleted to a certain level, at which point the service becomes fee-for-use until utilization drops again. By the same token, a service might use a linear or constant pricing algorithm until some threshold of resource use was reached, using an exponential strategy thereafter.

5.2.2 Service Differentiation

Service providers may want clients to receive different levels of access to resources or pricing based on the service category to which they belong. Service category in this case is an abstract group defined by the service provider that might relate to factors such as the client's service use profile e.g., "well-behaved" clients vs. "poorly-behaved" clients or some out-of-band, cached relationship e.g., "AOL customers" vs. education domain clients. Below are a few possible ways that a service might employ to differentiate between clients.

Buyer's Clubs: the price that a client pays depends upon the last price category it was in. Hence, people who paid for premium service might get a price break the next time they renew their subscription, or might receive a price break if they purchase a large enough unit of service.

Threshold-based Packet-Dropping: service categories such as gold, silver, and bronze correspond to certain price thresholds and packets for a particular subscription are dropped whenever the price rises above the level of service that the client paid. For example, if a client purchases "silver" service, packets are dropped at some fixed rate whenever the price is above a threshold.

Queuing Manipulation: level of service corresponds to the delay that the client experiences in having its packets processed.

Subscription Parameter Modification: the level of service corresponds to a certain set of values for subscription parameters. Example: Subscriptions are sold by packet. If a client purchases gold service, it is able to receive 50 packet blocks; silver = 30 packet blocks; bronze = 10 packet blocks.

Further research is necessary to determine how to combine different dynamic pricing mechanisms in order to enhance overall system survivability by discriminating against different kinds of adversarial behavior. However, our preliminary results indicate that dynamic pricing strategies offer a promising new direction in countering server-directed DDoS attacks on the Internet.

6. Bibliography

[BBC01]http://news.bbc.co.uk/low/english/sci/tech/newsid 1348000/1348820.stm

[CAIDA01] www.caida.org/outreach/papers/backscatter/

[Jue99] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks," Network and Distributed System Security Symposium '99, San Diego, CA, USA, February 1999.

[Ful00] E. Fulp and D. Reeves, "A Multi-Market Approach to Resource Allocation," Proc. of Networking 2000, Lecture Notes in Computer Science, G. Pujolle, ed., No. 1815, pp. 945-956, May 2000.

[Bla98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "RFC 2475: An Architecture for Differentiated Services," December 1998.

[Gib01] S. Gibson, "The Strange Tale of the Denial of Service Attacks against GRC.com", www.grc.com/grcdos.html", June 2001.

[Fis99] P. C. Fishburn and A. M. Odlyzko, "Competitive Pricing of Information Goods: Subscription Pricing Versus Pay-Per-Use," Economic Theory, Vol. 13, pp. 447-470, 1999.

[Niv73] L. Niven, "Flash Crowd," The flight of the horse, Ballantine Books, 1973.

[Bak99] Y. Bakos and E. Brynjolfsson, "Bundling Information Goods: Pricing, Profits, and Efficiency," Management Science, December 1999.

[Dwo92] C. Dwork and M. Naor, "Pricing via Processing or Combating Junk Mail," in Ernest F. Brickell, ed., Crypto '92, Vol. 740, Lecture Notes in Comp Science, pp. 139-147. Springer-Verlag, 16-20 August 1992.

[Riv97] R. Rivest and A. Shamir, "PayWord and MicroMint: Two Simple Micro-payment Schemes, " Lecture Notes in Comp. Science, vol. 1189, Proc. Security Protocols Workshop, Springer-Verlag, pp. 69-87, 1997.

[Jak99] M. Jakobsson and A. Juels, "Proofs of Work and Bread Pudding Protocols," In B. Preneel, ed., Communications and Multimedia Security. Kluwer Academic Publishers, pp. 258-272, 1999.

[Cha83] D. Chaum, "Blind Signatures for Untraceable Payments," Advances in Cryptology--Crypto '82, Springer-Verlag, pp. 199-203, 1983.

[Sno01] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, Proc. ACM SIGCOMM 2001, August 2001.

[MPAPI] W3C Micro-payments API and Markup WGs, www.w3.org/ECommerce/Micro-payments/

[Milli] Compaq Millicent, www.millicent.digital.com

[Pay] PayPal, www.paypal.com

[NS2] NS-2 simulator, http://www.isi.edu/nsnam/ns

[Arq] The ARQoS Project, http://arqos.csc.ncsu.edu/

[Free] The FreeHaven Project, http://www.freehaven.net/

[Digi] DigiCash, http://www.digicash.com

[Hash] A. Back, "Hash Cash: A Partial Hash Collision Based Postage Scheme," www.cypherspace.org/~adam/hashcash.

[**Mojo**] Mojo Nation, http://www.mojonation.net/