Preventing Denial of Service Attacks on Quality of Service

Errin Fulp[1], Zhi Fu[2], Douglas S. Reeves[2], S. Felix Wu[3], and Xiaobing Zhang[4]
[1] *Dept. of Computer Science, Wake Forest University*
[2] *Depts. of Computer Science and Electrical and Computer Engineering, N. C. State University*
[3] *Dept. of Computer Science, U. of California at Davis*
[4] *Ericsson*
fulp@wfu.edu, {zfu, reeves}@eos.ncsu.edu, wu@cs.ucdavis.edu, xzhang2@eos.ncsu.edu

Abstract

Capabilities are being added to IP networks to support Quality of Service (QoS) guarantees. These guarantees are needed for many applications such as voice and video transmission, real-time control, etc. Little attention has been paid to making these capabilities secure; in their present form they are vulnerable to attack. The ARQoS project is examining these vulnerabilities, and ways to prevent denial of service attacks on Quality of Service capabilities. In this paper, we describe two important parts of the project.

The first part is the application of a pricing paradigm to resource allocation. User acquisition of network resources must be authorized, and the relative amount of resources that can be requested is carefully controlled. We present a distributed method of pricing which is highly flexible and responsive to changing conditions. Experimental results illustrate its effectiveness.

The second part is the detection of TCP dropping attacks by compromised routers. The detection occurs at the end system and does not require any cooperation from the network. We have enhanced a method of statistically analyzing traffic patterns to detect dropping attacks. The method has been implemented and tested over the Internet; results are presented.

1. Introduction

Quality of service (*QoS*) is an emerging capability for IP networks. The usual components of QoS are an upper bound on the end-to-end delay and the delay variation (or *jitter*), and an upper bound on the packet loss or error rate. We believe that another dimension of QoS should be security (prevention and/or detection of attacks), which is the focus of the ARQoS project. (ARQoS is a play on the name "Argus", who was a creature from Greek mythology assigned by the gods to be a watchman.)

The addition of QoS will mean that IP networks can support a much broader range of such applications as voice and video transmission (both interactive and non-interactive), real-time distributed simulation and control, collection of data from sensors, and others. There are powerful incentives for supporting these applications and converging on a single networking infrastructure, including efficiency and cost, ease of management, accelerated deployment of the latest commercial technology, and the potential for creating applications that integrate communication and computing. Many of these benefits are relevant in a military environment.

Providing QoS requires network support at the packet level, and at the connection level, where a *connection* is the entire sequence of packets transmitted by an application to a receiver or receivers. Packet-level functions include scheduling / multiplexing, traffic shaping or smoothing, policing, packet dropping, and congestion control, while connection-level functions include signaling, admission control, routing, and resource reservation.[1] In our context, a *network resource* can mean link or switch bandwidth, buffer space, scheduling priority, server access, etc.

Figure 1 shows a typical IP network configuration that might support QoS. In this figure, a distinction is made between *access networks*, which have lower bandwidths and traffic volumes, and *core networks*, which have much higher bandwidths and must handle much greater traffic volumes. Because of this distinction, different protocols and technologies are used in access and core networks. For access networks, the Resource Reservation Protocol (or *RSVP*) [2] has been proposed as the fundamental means of ensuring QoS, while in the core networks, the major protocol is Differentiated Services, or *DiffServ*.[3]. RSVP is a means of reserving resources for individual connections (also called *microflows*), and can provide very strong guarantees of resource availability and QoS.

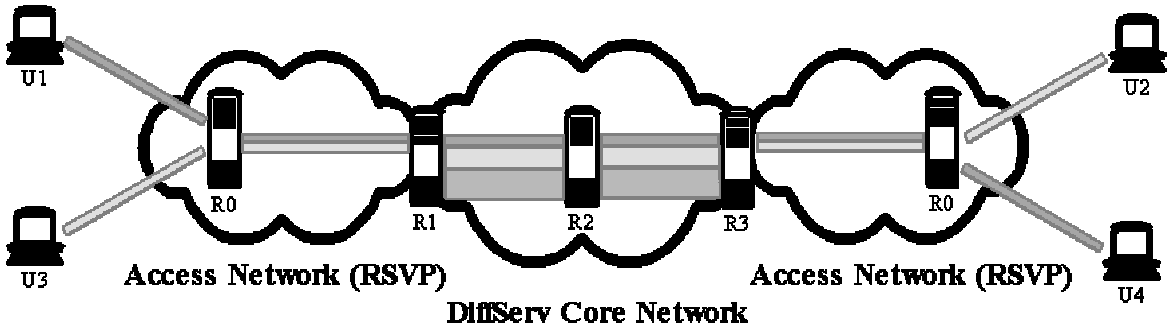


Figure 1: IP network supporting QoS

DiffServ aims to allocate resources and control QoS for aggregations of connections (also called a *flow*), and as such is considered to require less overhead and be more scalable than RSVP.

The importance and accessibility of the network infrastructure will soon make it a tempting target for attack. In light of that, it is wise to regard any new or proposed network capability as a potential vulnerability, and to design in security at an early stage. This has not been the case for QoS mechanisms, however, and must now be addressed. We classify vulnerabilities and attacks into two categories:

- Those acting at the connection level to interfere with the functions mentioned above (signaling, admission control, etc.). We refer to these as attacks on the *control flow*.
- Those acting at the packet level to interfere with the functions of scheduling, shaping, packet dropping, etc. We refer to these as attacks on the *data flow*.

The methods for responding to attacks can be categorized as prevention, detection, counter-attack, or recovery. The goal of the ARQoS project is to prevent and/or detect control and data flow attacks on QoS mechanisms. Our target network technology is the Internet. The project has many parts or sub-goals. In this paper, we focus on two of those parts. One of them, *resource pricing*, is intended to prevent attacks on the control flow. The other, *analysis of TCP dropping attacks*, is aimed at detecting attacks on the data flow. These two major parts will illustrate the breadth of the project, important ideas behind our work, and the progress made so far.

2. Resource pricing

A necessary step in providing QoS is allocating network resources to meet the needs of applications. There are many ways in which this process can be subverted. As an example, if there is no authorization process in place, and no means for controlling the allocation amount, any user may request and reserve any amount of resources they wish. Another example is to forge, delete, or illegally modify a reservation message so that an unauthorized amount of resources is received, or access is denied to an authorized user. In this section, we address the first issue of authorizing and controlling resource allocation via *resource pricing*.

A set of users wishes to share a resource (or resources). The amount of the resource has been previously fixed, and there may be an insufficient quantity of the resource for every user to get as much as desired. In this case, a method of allocating the resource to satisfy some external goal must be devised. To this end, a price per unit of resource is calculated and made known to the users. Each user u has a fixed budget b_u with which a resource r at price p_r per unit can be purchased. The user can afford at most b_u / p_r units of resource at this price. If resources cannot be stored and sold at a later time, an obvious goal is to set the price low enough such that as much of the resource will be sold as possible at the time it is available. A budget can be thought of simply as an indication of the relative resource amounts that users may obtain; all other things being equal, a user with twice as large a budget as another will be able to obtain twice as much resource. While the means of determining and distributing user budgets is outside the scope of our work, we note that it could be based on many factors, including the ability to pay, urgency or importance of function, degree of trust, etc.

Our method is based on the notion of *demand-based pricing*. The components of the method are 1) measurement of demand; 2) price calculation; and 3) price distribution. Since resource demand is based on price, and price is calculated from demand, a feedback system results. This process can be iterated to reach an equilibrium point that has desirable properties, such as satisfying a target resource utilization. The iterative measurement of demand, followed by computing a new

price, etc., until equilibrium is reached, is termed a *tatonnement process* [4]. In our work, the tatonnement process has the following form:

$$p_r^{i+1} = p_r^i \cdot d_r^i / (\alpha \cdot s_r)$$

where p_r^i and p_r^{i+1} are the price of resource r in the i^{th} and $i+1^{th}$ iterations, respectively, d_r^i is the measured demand for resource r during the i^{th} iteration, s_r is the supply of resource r, and α is a constant less than 1. The purpose of α is to increase prices before utilization reaches 100%. It can be shown that a tatonnement process will result in an allocation that is *Pareto-optimal*. An allocation is Pareto-optimal if no user can get a greater amount of resource without another user receiving a smaller amount of the resource.

When there is one user u with budget b_u and one resource r available in quantity s_r, the simplest possible case, the equilibrium result is a price p_r such that demand for the resource by that user equals the supply. When a set of users U compete for a single resource r, and each user $u \in U$ has budget b_u, at equilibrium the total demand equals the supply, and users are allocated resource amounts in proportion to their budgets. The budgets can be viewed as *weighting factors* in allocating the resource among the users.

Now consider the case where the users desire *multiple* resources. In our discussion, we assume that user u demands the same amount of each resource. This is the case for bandwidth allocation, for example, where a connection requires the same bandwidth on every link it traverses. Let R be the set of all resources, and R_u be the set of resources demanded by user $u \in U$. Then a tatonnement process may be executed independently for each resource r to reach an equilibrium price for that resource.

Let r'_u be a resource in R_u whose equilibrium price is greater than or equal to that of any other resource in R_u. r'_u is called a *limiting resource* for user u. Given that the user has one budget b_u, but faces $|R_u|$ resource prices, one for each resource it needs, how is the budget to be allocated? The answer to this question determines the type of *fairness* that the resource allocation policy implements. One important form of fairness, frequently advocated for network congestion control, is *max-min fair*: users who are resource-limited by the same resource will share in that resource equally. A more general form of fairness is *weighted max-min fair*; if users u and v have weights w_u and w_v respectively, and r' is a limiting resource for both u and v, then the ratio of resources allocated to u and v is equal to w_u / w_v. We can achieve both max-min fairness and weighted max-min fairness with pricing, in the following way. Each resource executes an independent tatonnement process to compute its equilibrium price. A user u for which r'_u is a limiting resource is allocated $b_u / p_{r'}$ of every resource $r \in R_u$. Thus, the price charged to u is equal to the price of the most expensive resource u requires, and the amount of resource allocated is proportional to u's budget (i.e., its weight). Under these conditions, it can be shown the system will result in a weighted max-min fair allocation; when the budgets of all users are the same, the allocation is simply max-min fair.

Another important form of fairness is *proportional fairness*. Suppose the relative change for a user u between one set of resource prices and another is equal to the change in its allocation amount, divided by the allocation amount (i.e. if u can afford 4 units of resource under one set of prices and 5 units under another, the relative change is (5-4)/4 = .25). Informally, a price assignment is proportionally fair if the sum of the relative changes for all users between this price assignment and any other price assignment is less than or equal to 0. Similar to max-min fairness, there is a weighted version of proportional fairness. Proportional fairness is important because, among other things, it is the form of fairness exhibited by TCP congestion control. If we compute prices for each resource independently, but allocate resources such that user u is allocated $b_u / (\Sigma_{r \in Ru} p_r)$ then it can be shown that the resulting allocation is weighted proportionally fair. A formal statement of these properties, and their proofs, may be found in [5].

A common notion in economics is the *utility curve*. A utility curve is a function relating resource allocation to the degree of utility or satisfaction experienced by the user. There are many forms of utility, but a common case is that utility is monotonically non-decreasing in allocation amount. It is possible and even likely for each user to have a unique utility curve representing that user's particular valuation of resources. For example, a user using a network for voice transmission would normally be satisfied with a much lower resource allocation than a user who was transmitting a video. Two other forms of fairness may now be defined. An allocation is *equitable* if users who are resource-limited by the same resource enjoy the same degree of utility, i.e., are equally satisfied with their allocation. Secondly, an allocation is *utility-maximizing* if it results in the maximum aggregate utility (sum of the utilities of the users of the network) of all possible allocations. It can be shown that our pricing method can accomplish either of these fairness goals. Again, details may be found in [5].

We have described above how prices are calculated. Other important issues are how to measure demand, since the tatonnement process uses that as an input, and how to distribute prices to the users. The tatonnement process itself is a simple computation, so the speed with which equilibrium is achieved will depend mainly on the number of iterations required, and the time required to distribute prices and measure the change in demand.

Convergence time is important in an environment where demands and/or resource availability can change frequently and rapidly, which is the case in computer networks. We address these issues below.

There are many benefits of the pricing approach that has just been presented. Foremost among them is flexibility. Using one mechanism for budget assignment, price adjustment, demand measurement, and price distribution, we can implement an abundance of different allocation policies. Ours is the most flexible method known in this regard. We believe this is a major benefit for network operation in a variety of different environments. In the section below on experimental results, we demonstrate that our pricing method is also fully distributed (scalable), is robust, adapts quickly to changes, has low overhead, and results in efficient resource usage.

Our work is based on results from microeconomics [6]. There have been other proposals to apply microeconomics to allocation of network resources, including [7] [8] [9] [10] [11]. The differences of our work with those include:

- Our method regards fairness as a policy; we present a mechanism that can support a variety of fairness goals, or policies.
- Our method is fully distributed and has been shown to converge quickly under dynamic and realistic network conditions.
- Our method makes no assumptions about the type of traffic allowed, and supports a more general model of user behavior.
- As discussed below, our model is the only one to support both reserved and dynamic resource pricing.

2.1. Implementation and experimental results

Pricing can be used to dynamically determine the appropriate transmission rate for connections using a network. For ATM networks, this is the purpose of ABR rate control, while for IP networks, it is the function of TCP congestion control. We simulated rate control for ATM networks using our pricing mechanisms; our implementations, and the results, are presented here.

Traffic loads for the ATM ABR class may be frequently changing as new connections start, old connections stop, and the transmission requirements of existing connections change (due, for instance, to the variable bit rate output by a codec). A method of pricing must converge quickly to an equilibrium value under these circumstances. Our first experiment simulated a set of 20 users competing for a single resource. All user budgets were equal, but the user demands were randomly and uniformly distributed. Starting from an initial price, the tatonnement process was executed until equilibrium was achieved. This process was repeated 10,000 times; the results are summarized in Figure 2. The x-axis in this figure measures the ratio of the initial price to the equilibrium price, and varies from .01 to 100. The y-axis indicates the number of iterations required for convergence; the average and 95% confidence intervals are shown. From this example it may be seen that convergence occurs very quickly, typically requiring only a few iterations.

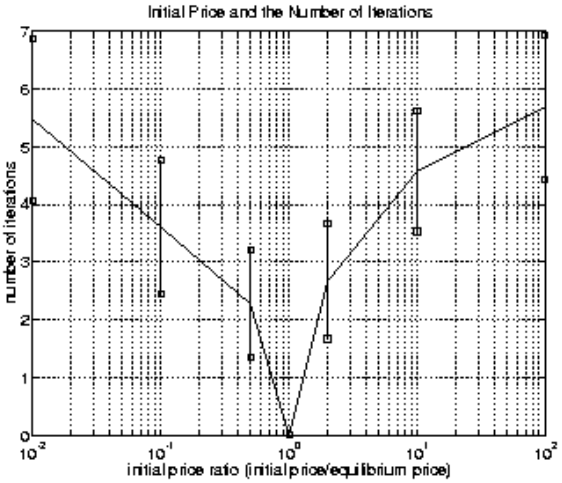


Figure 2: Dependence of convergence time on the initial price estimate

Our second experiment investigated the ability of our pricing method to achieve fair allocations in a complex network, with rapidly varying traffic conditions. For an experimental network, we used a benchmark proposed by the ATM Forum [12] for investigating resource allocation methods. It models substantial competition between users with differing routes and widely varying propagation delays. The traffic source for each user was taken from a set of actual MPEG VBR video traces. There were two groups of users; budgets for user group #1 were set to twice the budgets of users in group #2. Traffic demand on each link was measured and a new price computed every 10 ms. Prices were distributed to the users using ATM RM cells (already standardized for explicit rate allocation). We are interested in the utilization of link bandwidth, and the fairness of the allocation. The fairness of the allocation is measured according to the *fairness index*, which measures how far from optimally fair an allocation is. A measurement of 1.0 indicates complete fairness, while 0.99 has been proposed as the appropriate definition of "fair".

The results are shown in Figure 3. In the first case, the goal of the price computation was a weighted max-min

fair allocation; in the second case, the goal was a weighted proportionally fair allocation. From these results, it may be seen that utilization was very high, and the allocation was *always* fair. These results are extremely encouraging, given that they incorporate the characteristics of real networks and actual traffic. Details may be found in [5]. In summary, advantages are:

- The price calculation and allocation is fully distributed. This bodes well for scalability and robustness. There is no synchronization among users, or in the computation of resource prices.
- Overhead is low, involving demand measurement, a simple iteration, and distribution of prices. The distribution of prices in the example above would require less than 1% of the bandwidth of the data traffic.
- Efficient and fair resource utilization.
- Rapid adjustment to changing conditions.

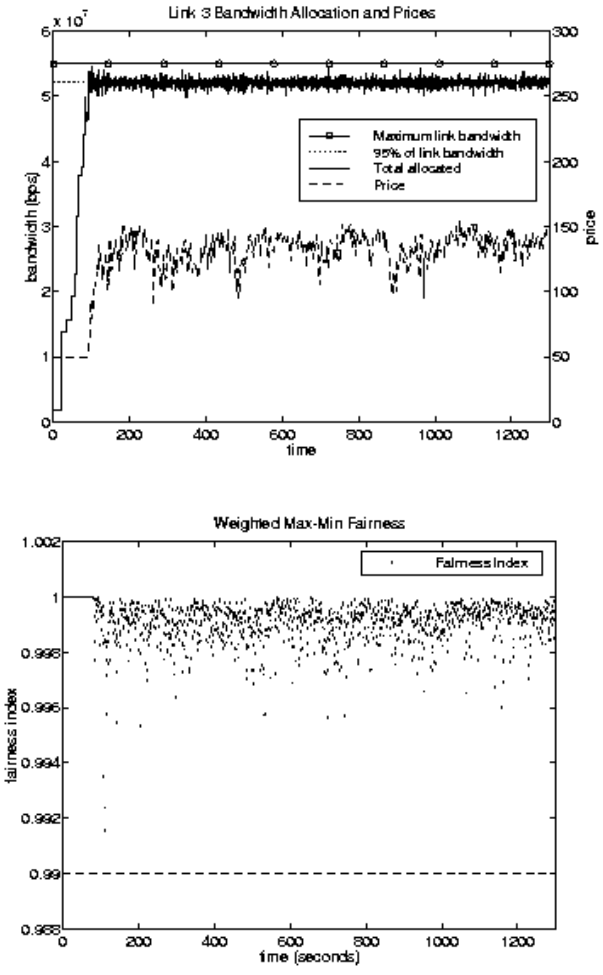


Figure 3a: Weighted max-min allocation

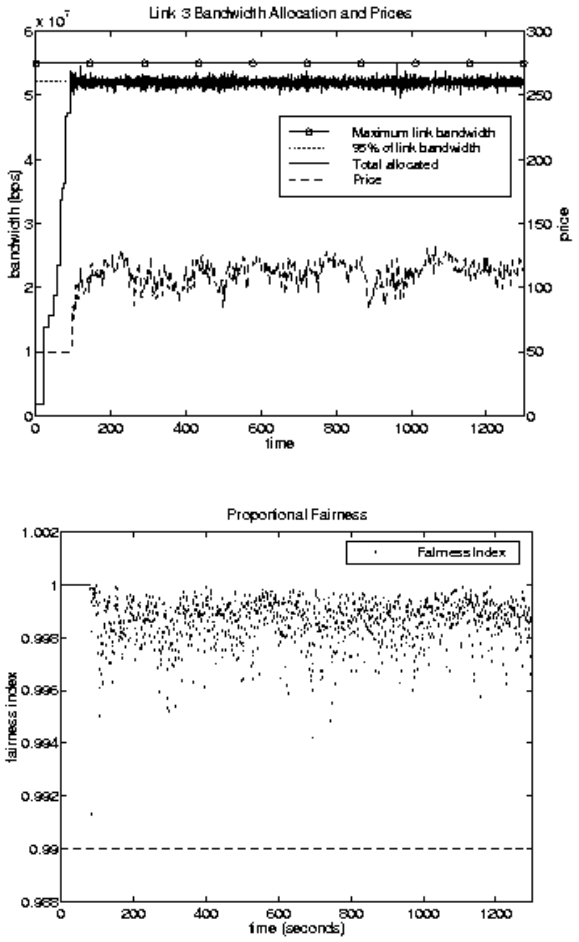


Figure 3b: Proportionally fair allocation

2.2. Pricing of reserved resources

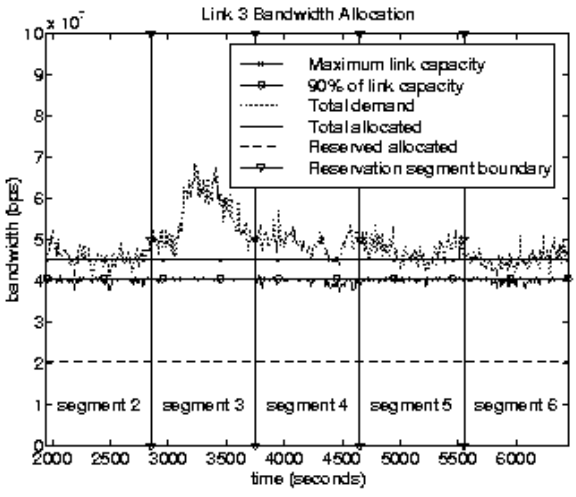
One shortcoming of the pricing method as presented is that while utilization, fairness, and resource access are stable, prices are not. This means the allocation per user changes dynamically, as is normally the case with reactive congestion control. Applications desiring a stable resource allocation may not be able to tolerate such fluctuations in price and resource allocation amount.

To deal with this problem, prices must be computed over longer intervals of time. A demand-based price computation must then address the issue of predicting demand over an interval. There are numerous bases for predictions, including past measurements of demand (possibly over long periods of time), current demand, and auctions. Prediction of demand is outside the scope of our project, but implementing stable pricing is within scope. The result of fixed prices is that resources are reserved for use once they are acquired. New users may thus find they are denied admission to the network

because there are insufficient resources not already reserved.

In our work, we are interested in a two-price model, where one set of prices are based on predicted demand, and another set of prices are based on current (instantaneous, or immediate) demand. Prices of resources whose demand is predicted will remain stable for the specified interval, while those based on current demand are free to vary. Users wishing to acquire resources may do so from either pool. The preference of users for resources with stable prices over resources with varying prices can be expressed by means of an *indifference curve*, a common concept in economics [6]. This allows optimization of resource allocation according to the preferences of users. In general, it should be expected that prices for reserved resources will be higher than those for variably-priced resources.

We implemented the two-price model on the same ATM network used in the previous experiment. Reserved prices were computed based on predicted demand, and any resource not actually purchased at the reserved price was available at a price based on the immediate (and varying) demand. Users were categorized as those who prefer *reserved* prices, or those who *prefer cheaper* prices. The results are shown in Figure 4. From this figure, it may be seen that utilization is high, the allocation method is responsive to changing demands and reaches equilibrium quickly, and the price of reserved resources is normally less than the price of "spot" resources (those computed from immediate demand).



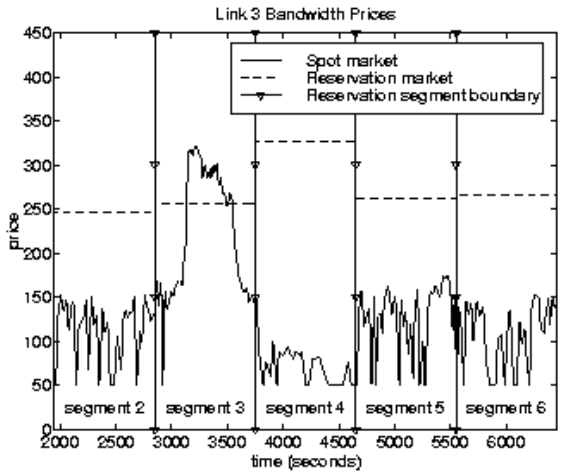


Figure 4: The two-price model.

This version of pricing (that supports the two-price model) supports users with varying preferences for stability vs. maximum resource amount. We discuss next an application in which reservations are desirable.

2.3. Application to RSVP

Pricing is an appropriate method of resource allocation for individual connections when those resources are reserved in advance. As mentioned, RSVP is a well-known method of resource reservation for individual connections. Figure 5 depicts a typical configuration for an RSVP-enabled network. In this diagram, a reservation message from source to destination is transmitted using RSVP. At each router, this message is intercepted and a policy request is made using the COPS [13] protocol. The COPS request is handled by a policy server and an admission control decision is made by that server. If the decision is positive, the response to the router indicates that resources should be reserved for the requesting connection, and the RSVP reservation message is propagated onwards to the next router. If the decision is negative, the response from the policy server is propagated to the user to indicate that admission is denied, and no reservation is established. The vulnerability in this scheme is that there is no incentive for users to limit their requests. The policy server lacks the information needed to distinguish or discriminate between users, and to allocate resources fairly when they are scarce. The result is that reservations can easily be used to acquire (and waste) resources unnecessarily.

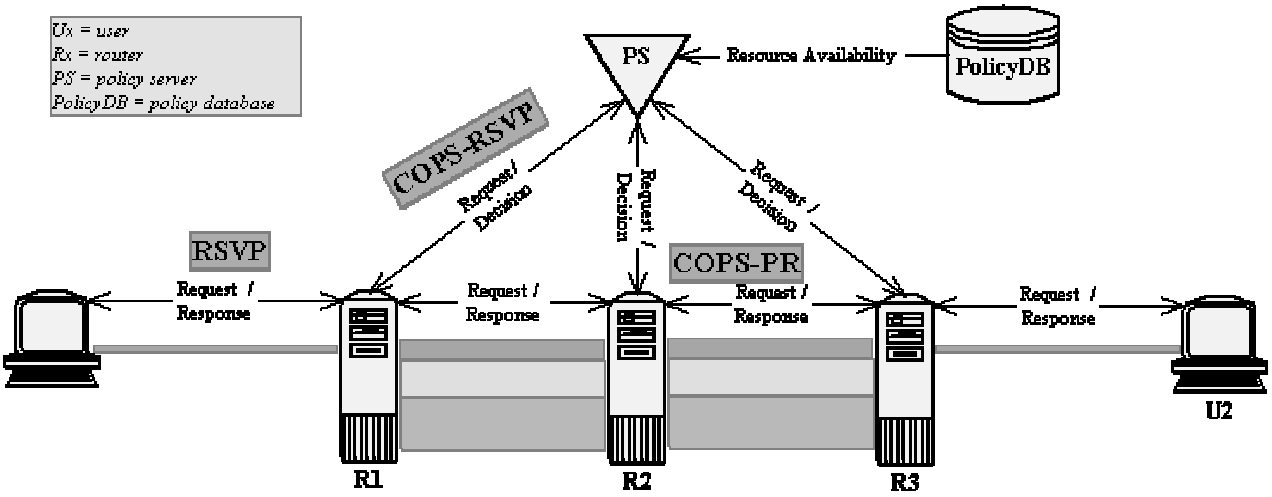


Figure 5: RSVP / COPS interaction

We propose to use pricing to improve this situation. The protocol changes we are proposing and implementing are illustrated in Figure 6.

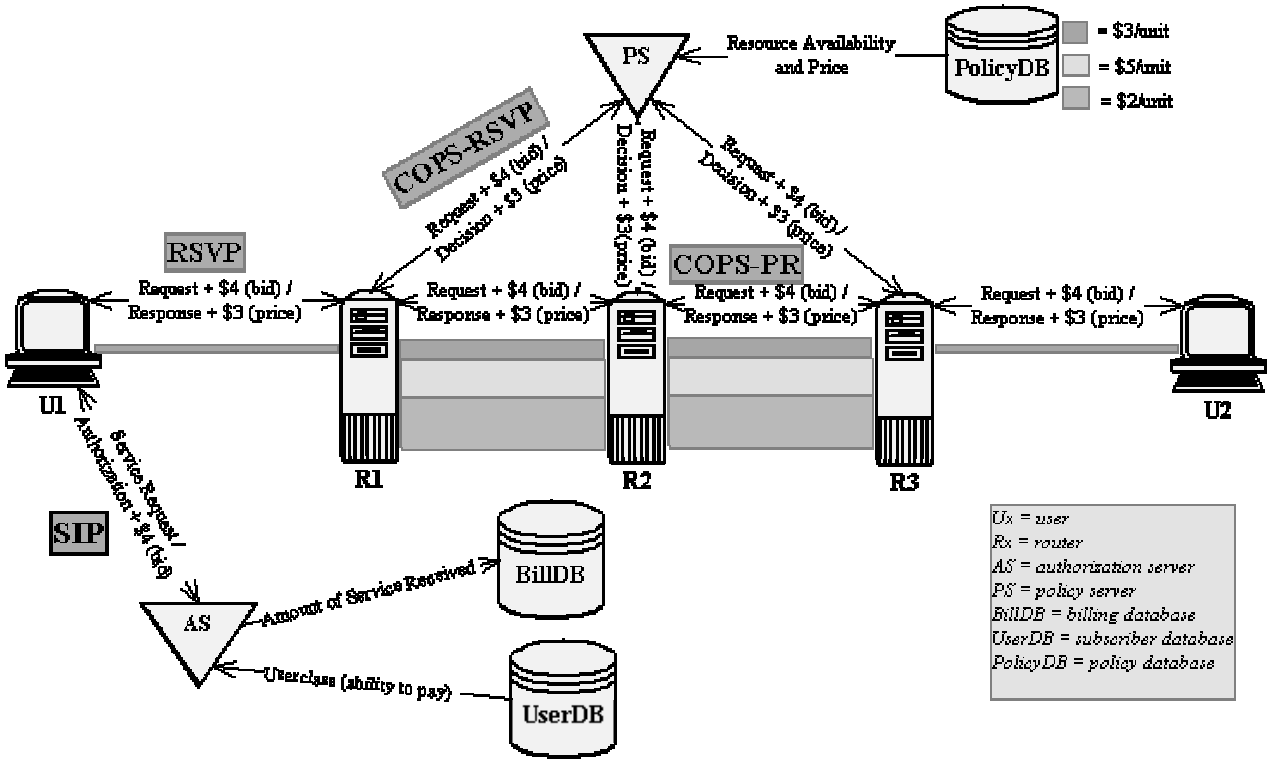


Figure 6: RSVP/COPS and other infrastructure for pricing

The changes we propose are as follows:

1. A user wishing to reserve resources contacts an authorization server to receive authorization. We suggest the use of SIP (Session Initiation Protocol) for this purpose. The authorization server responds with a *signed policy object* that verifies the ability of the user to pay a specific price.
2. This signed policy object is included in the RSVP message by the sending application.
3. When the RSVP reservation message is intercepted, the policy object is copied into the COPS request message. The policy server can then consider this information, and current resource prices, in making the admission control decision.
4. The response to the user includes the price being charged for the resource. The user can propagate this information back to the authorization server if desired.

The changes required are generation at the connection server of signed policy objects, attachment of the object to RSVP and COPS messages, and attachment of the price to the policy server's response. We are currently investigating which signature mechanism is the best choice in terms of scalability and security. The major benefit of the proposed approach is that it allows a high degree of independence between the connection server and the policy server. Communication between them is conducted using existing protocols, and neither must know the identity or location of the other.

We believe the work done on pricing is a valuable part of preventing denial of service on quality of service mechanisms. The method we have presented is highly flexible, and is practical to use in large networks. Most importantly, perhaps, is that it gives incentives to users to user resources responsibly, and provides disincentives for fraudulent use of resources.

3. Detection of TCP packet dropping attacks

Among the various types of denial of service (*DoS*) attacks, the *packet dropping attack* is one of the most difficult to handle. As part of the ARQoS project we have studied the impact of packet dropping attacks on network quality of service, and methods of detecting such attacks. A *compromised* router (one which has been hacked) can choose different dropping patterns to degrade TCP service by varying degrees; selectively dropping a very small number of packets can result in severe damage to TCP performance. Three packet dropping patterns have been investigated: periodic, retransmission-based, and random. We describe a statistical analysis module (a *SAM*) for the detection of TCP packet dropping attacks. This analysis module focuses on how to effectively distinguish normal packet dropping (due to TCP behavior [15]) from malicious dropping. The metrics of average packet delay (the *delay metric*), the position or sequence number of reordered packets (the *position metric*), and the number of packets reordered (the *number metric*), have been used for purposes of detection. We have evaluated and compared the effectiveness of this approach through experiments involving four FTP sites accessed across the Internet.

3.1. Dropping Patterns

The victim under a particular dropping attack may be any TCP connection; in our experiments, we have studied only FTP connections. An FTP connection that is attacked is called a *victim connection*, and packets dropped by an attacker are called *victim packets*. We classify packet dropping patterns as follows.

1. **Periodic packet dropping (PerPD)**: Packets are periodically dropped in a connection according to 3 parameters symbolized by *K*, *I*, and *S*. *K* is the total number of victim packets in the connection. *I* is the interval between two consecutive victim packets, and *S* is the position of the first victim packet in the connection. In this pattern, every packet is counted, including retransmitted packets.
2. **Retransmission packet dropping (RetPD):** In this attack, the attacker always drops the retransmissions of a specific packet. Two parameters *K* and *S* are defined for this pattern. *S* denotes a specific victim packet. *K* denotes the number of times this packet (including retransmissions) is dropped. When a retransmitted packet is dropped, TCP retransmits again only upon expiration of the retransmission timer, and enters the slow start phase of operation. For each retransmission, the retransmission timeout value (RTO) is doubled, with an upper limit of 64 seconds. This doubling is called exponential backoff. We can infer that after a few consecutive unsuccessful retransmissions, the sender has to wait a long period of time before attempting a new retransmission.
3. **Random packet dropping (RanPD):** Attackers randomly choose up to *K* packets to drop in a connection. Since this kind of attacks behaves more like normal packet dropping in the Internet, it may be expected that the fast retransmit and recovery mechanism [15] of TCP will work effectively in response to packet loss. From this point of view, we would expect that random dropping has limited impact on TCP's performance, compared with other dropping patterns.

3.2. Empirical study of impacts of packet dropping attacks

In order to investigate the impact of packet dropping attacks, we performed a series of experiments using different dropping patterns. All experiments were based on FTP bulk data transmission from servers to clients. File transfers were conducted between 4 client-server pairs in the Internet; the client was located in our lab at N. C. State University, while the servers were located in the U.S., Europe, and Asia (see Table 1). The data file used for transfer was an identical, mirrored file of size 5.5 MB.

Table 1: FTP sites used in the experiments

Name	FTP Server / IP Address	Location
Heidelberg	ftp.uni-heidelberg.de 129.206.100.134	Europe
NCU	ftp.ncu.edu.tw 140.115.1.71	Asia
SingNet	ftp.singnet.com.sg 165.21.5.14	Asia
UIUC	ftp.cso.uiuc.edu 128.174.5.14	North America

The client ran under Linux 2.0.36 with IP firewall and divert socket support. In order to simulate an attacker, a program named attack-agent was executed on each client. For every arriving FTP data packet, the attack-agent captured it through a divert socket, and then released or dropped the packet according to a predefined dropping pattern.

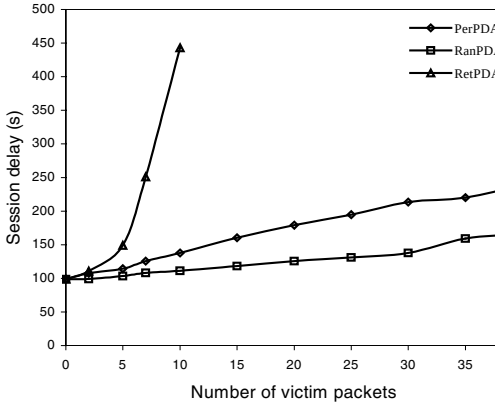


Figure 7: Connection delay for the NCU site under three dropping patterns

Figure 7 evaluates the impacts of three dropping patterns for the NCU site (National Central University in Taiwan). Other sites show similar behavior under these attacks. It is clear from the Figure 7 that given the same number of victim packets, the retransmission packet dropping attack interferes the most with TCP performance.

3.3. Intrusion detection for malicious packet dropping

In this section, we describe the design and implementation of our statistics-based intrusion detection tool. This tool runs at the FTP client site, to determine whether a particular TCP flow is experiencing a malicious dropping attack or not.

SRI's NIDES/STAT [16] algorithm monitors a subject's (either a user or a software program) behavior on a computer system, and raises alarm flags when the subject's current (short-term) behavior deviates significantly from its expected behavior. The expected behavior is described by its long-term profile. The NIDES algorithm is based on a χ^2-like test for comparing the similarity between the short-term and long-term profiles. We now briefly describe this algorithm.

Let the current system behavior be a random variable under the sample space S. Events, $E_1, E_2 \ldots E_n$, represent a partition of S, where these n events are mutually exclusive and exhaustive. Let $p_1, p_2 \ldots p_n$ be the probabilities of occurrence corresponding to events $E_1, E_2 \ldots E_n$. The system behavior is sampled N times, where N is a large number. Let Y_i represent the number of occurrences of event E_i in these N experiments. Thus, we have $p_i = Y_i/N$, where $\sum_{i=1}^{n} p_i = 1$.

A short-term profile is taken from a smaller number of measurements N'. Y_i' is the number of occurrences of event E_i in these N' experiments, and $p_i' = Y_i'/N'$. To determine whether a short-term profile has a similar probability distribution with the corresponding long-term profile, the following hypothesis is tested [17]:

$$H_0 : p_i' = p_i, \quad i = 1, 2, ..., n$$

$$H_1 : H_0 \text{ is not true}$$

Define Q to be a measure of similarity, as follows:

$$Q = \sum_{i=1}^{n} \frac{(Y_i' - N' \times p_i)^2}{N' \times p_i}$$

Intuitively, Q measures the closeness of the observed (short-term) probability distribution to the expected (long-term) distribution. A small Q favors the hypothesis H_0, while a large Q favors H_1. If independence is assumed between events E_i, and the experiments are carried out independently, it has been

proved that, for a large N', Q has an approximate χ^2 distribution with n-1 degrees of freedom. To get an accurate approximation, it is suggested that N' should be larger than 50 and ($N' \times p_i$) should be larger than 5. Otherwise, several 'rare' events should be merged together to form a new event such that ($N' \times p_{new}$) will exceed 5, where p_{new} denotes the probability for the merged event.

Let q be an instance of Q, and α be the desired significance level of the test. If Prob($Q > q$) < α, or $q > \chi^2_{\alpha,(n-1)}$, the hypothesis is rejected. In the context of our application, it means that the short-term profile is statistically significantly different from the corresponding long-term profile, leading to the conclusion that anomalous behavior has occurred. In drawing this conclusion, two kinds of errors may occur. A Type I error means that the hypothesis is true, but was rejected according to the test. A Type II error means that the hypothesis is false, but was accepted as true by the test. The frequency of occurrence of Type I errors is also referred to as the false positive rate, while the frequency of occurrence of Type II errors is referred to as the false negative rate.

In practice, however, the assumption of independence between events may not hold true. As a result, Q may not have a χ^2 distribution. The NIDES/STAT algorithm proposes to empirically measure the probability distribution of Q. This distribution of Q values for the short-term measurements, along with the distribution of the system's long-term behavior (i.e., p_1, p_2 … p_n), is saved in a profile. This profile is updated per UPDATE_PERIOD (24 hours in NIDES) in a real-time operation.

Training is the process by which the statistical component learns the normal behavior for a subject. In NIDES/STAT, the long-term profile training consists of three phases: Category training (learning the subject's expected behavior, i.e., the probabilities of events, E_i.), Q training (learning the empirical distribution of Q through a sequence of short-term measurements) and threshold training (establishing an appropriate threshold for detecting anomalous behavior).

3.4. Experiments

We implemented a method of detecting packet dropping attacks which was based on the statistical analysis methods of SRI's NIDES/STAT. We call our method *TDSAM*, for TCP dropping statistical analysis module. In our method, three metrics — total ftp connection delay, the position of packet reordering, and the number of packet reorderings per connection — were used as input to the statistical analysis model for intrusion detection. We hypothesized that the connection delay, the position and number of packet reordering would all deviate significantly from their corresponding normal patterns [18] under a dropping attack. The reasons for using packet reordering rate vs. packet loss rate are that (1) dropping packets usually results in packet reordering, and (2) at the receiver side, packet losses are difficult to measure reliably. While a server (sender) can easily detect data packet loss by noticing a retransmission of the packet, a client (receiver) cannot be sure of a packet loss unless the sender informs it of the transmission. In our experiment, however, receivers did not have such information.

To detect an out-of-order packet, the receiver records the sequence number of the last-received non-reordered packet; this is denoted as Seq_{max}. For each arriving packet, if its sequence number is larger than Seq_{max} (with wrapping comparison), the packet is considered non-reordered, and Seq_{max} is set to the sequence number of this arriving packet. Otherwise, we test if the arriving packet has been received before. If not, it must be out of order. Otherwise, it is a redundant retransmission, possibly caused by an ACK loss, coarse feedback, or an invalid retransmission timeout. Note that we do not count this as reordering in our experiment. For instance, if 5 packets, P_1, P_2 … P_5, are sent in sequence and received in the order, P_1, P_2, P_3, P_5, P_4, then we say that P_4 is an out-of-order delivery.

For each FTP site, an experiment was conducted, consisting of 4 steps: (1) normal (non-attacked) long-term profile establishment, (2) normal (non-attacked) short-term Q distribution measurement, (3) measurement of short-term profiles while under attack (according to different dropping patterns), and (4) intrusion detection using our statistical analysis module, TDSAM. We now describe steps (1) and (2) in more detail, using just the position metric as an example.

Long-term Profile Establishment: We formed the long-term profile by running 20000 FTP connections consecutively. For each FTP connection, the observing window covered all data packets. The window was evenly divided into n bins, numbered from 0 to n-1. For example, if a connection transmitted 4000 packets and n=5, then the observing window ranged from 1 to 4000 and the bin-width = 800 packets, with bin_0 = [1, 800], bin_1 = [801, 1600], … and bin_4 = [3201, 4000].

In each connection, the position of reordered packets was measured. Let $Count_i$ represent the number of out-of-order packets whose sequence number fell into the i^{th} bin. We summed up the $Count_i$ for all the FTP connections and calculated the probability of each bin as follows:

$$p_i = Count_i / \sum_{k=0}^{n-1} Count_k \qquad 0 \le i \le n-1$$

***Q* Distribution Measurement:** A *Q* value was calculated by comparing the short-term profile with the long-term profile. The short-term data were collected from 5000 FTP connections. Information about an FTP connection was defined as a record, grouped into record sets. Multiple consecutive records were accumulated until a record set satisfied the requirements of a *Q* test. These requirements included that (1) the total number of records *N'* in the set should be larger than 50; (2) for each bin, the value, $N' \times p_i$, should be larger than 5. For each such record set, a *Q* value was calculated.

Q_{max} was defined as the maximum *Q* value calculated from any of the record sets. The distribution of *Q* values from 0 to Q_{max} was divided into 32 intervals, and the frequency of each interval was obtained. The resulting *Q* distributions are shown in Figure 8, for the case where 5 bins were used. We see that the last several intervals in the *Q* distribution have small tail probabilities. If a *Q* value falls into one of these intervals, this means that the corresponding short-term behavior deviates significantly from the normal behavior.

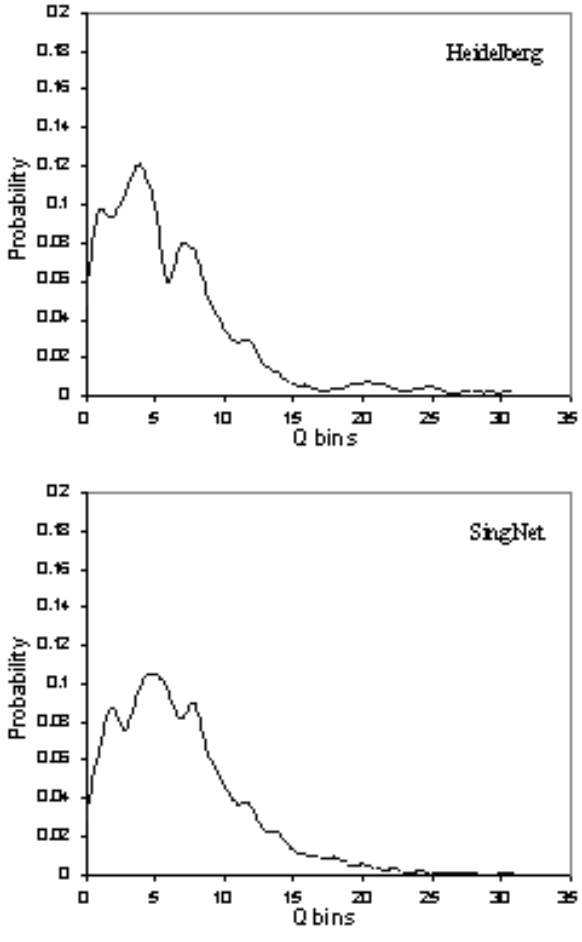


Figure 8a: Q distribution for position measure, when *nbin*=5

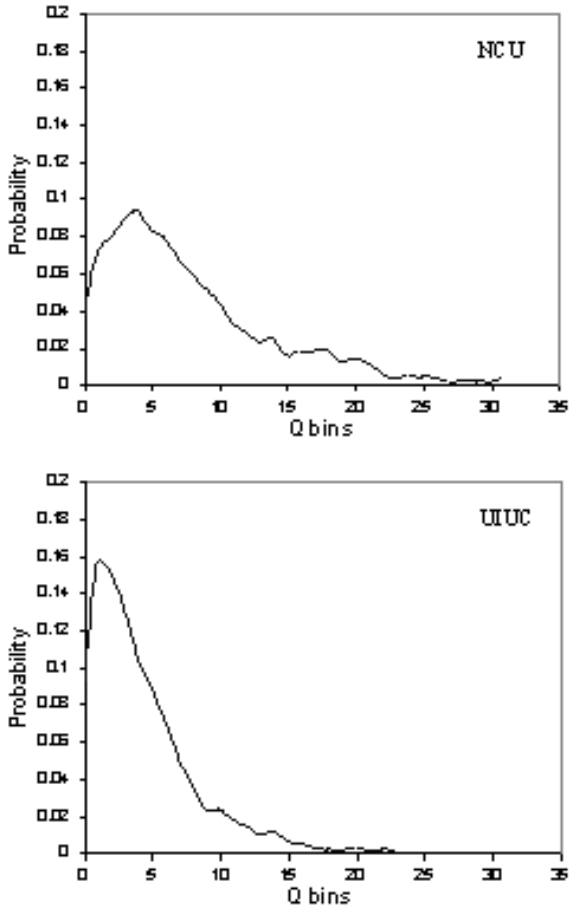


Figure 8b: Q distribution for position measure, when *nbin*=5

3.5. Results

After establishing the long term profile and measuring the *Q* distribution, we subjected the ftp connections to attack using our attack agent. An important value called the *red threshold* was defined for the *Q* distribution. The value of this threshold was set to .01 in our experiments. If a test generates a *Q* value whose probability is less than this threshold, according to the *Q* distribution, TDSAM will raise an alarm. This alarm indicates that an attack has been detected.

In Tables 2, 3 and 4, due to limited space, we only show the intrusion detection rate (*DR*). The "normal" case is one in which no attack in fact occurred. In this case, *DR* represents the false positive rate, and there are no false negatives. The other cases are different instances of attacks. In these cases, *DR* is the true positive rate, and (1-*DR*) is the false negative rate.

Table 2 summarizes the intrusion detection results using the position metric (position of reordered packets), Table 3 shows the results using the number

metric (number of reordered packets), and Table 4 shows the results using the delay metric (average packet delay). In these tables, the parameters of the periodic (PerPD) attacks are the total number of dropped packets, the spacing between dropped packets, and the position of the first packet to be dropped. For the retransmission (RetPD) attacks, the parameters are the number of times one packet will be dropped, and the position of that packet. For the random (RanPD) attacks, the sole parameter is the total number of dropped packets. The parameter *nbin* refers to the number of bins used in the long-term measurement.

The results show that TDSAM has a high detection rate for most periodic dropping attacks. This is because normal packet reordering nearly uniformly distributes across a connection, but periodic dropping attacks usually generate an extremely different packet reordering distribution. For instance, the long-term profile of the Heidelberg site is: p_1=0.194339, p_2=0.200759, p_3=0.197882, p_4=0.204260, p_5=0.202760, where *nbin*=5 and bin-width = 800. The PerPD (20, 4, 5) attack drops packets only in the first 85 packets, resulting in a large number of occurrences of packet reordering in the first bin. Under the attack, the corresponding distribution reordering becomes: p_1=0.837264, p_2=0.039390, p_3=0.043192, p_4=0.041045, p_5=0.039109.

Table 2: Detection rates for the position metric

Position *nbin*=5	Dropping pattern	Heidel-berg	NCU	Sing-Net	UIUC
Normal*	-	4.0%	5.4%	3.5%	6.5%
PerPD	(10,4,5)	99.7%	100%	100%	100%
	(20,4,5)	100%	98.1%	99.2%	100%
	(40,4,5)	96.6%	100%	100%	98.5%
	(20,20,5)	100%	100%	100%	100%
	(20,100,5)	98.9%	99.2%	99.6%	99.1%
	(20,200,5)	0%	76.5%	1.5%	98.3%
	(100,40,5)	0.2%	0%	0%	100%
RetPD	(5,5)	84.9%	81.1%	94.3%	97.4%
RanPD	10	0%	42.3%	0%	0%
	40	0%	0%	0%	0%
Intermittent (10, 4, 5)	5	98.6%	100%	98.2%	100%
	50	34.1%	11.8%	89.4%	94.9%

Table 3: Detection rates for the number metric

Position *nbin*=10	Dropping Pattern	Heidel-berg	NCU	Sing-Net	UIUC
Normal*	-	5.6%	6.1%	2.8%	4.5%
PerPD	(10,4,5)	100%	100%	100%	100%
	(20,4,5)	99.2%	98.7%	100%	100%
	(40,4,5)	97.3%	100%	99.6%	100%
	(20,20,5)	100%	100%	100%	100%
	(20,100,5)	100%	100%	98.9%	99.6%
	(20,200,5)	65.1%	90%	0.6%	99.1%
	(100,40,5)	4.5%	0%	0%	100%
RetPD	(5,5)	97.5%	94.4%	95.7%	100%
RanPD	10	0%	76%	0%	19.1%
	40	0%	0%	0%	18.2%
Intermittent (10, 4, 5)	5	100%	100%	100%	100%
	50	52.9%	22.1%	71.8%	96.2%

Table 4: Detection rates for the delay metric

Position *nbin*=20	Dropping pattern	Heidel-berg	NCU	Sing-Net	UIUC
Normal*	-	15.6%	9.0%	1.6%	5.8%
PerPD	(10,4,5)	100%	100%	100%	100%
	(20,4,5)	100%	97.5%	98.3%	99.2%
	(40,4,5)	96.4%	100%	100%	100%
	(20,20,5)	100%	100%	100%	100%
	(20,100,5)	100%	99.1%	99.2%	100%
	(20,200,5)	0.7%	83.2%	0%	99.5%
	(100,40,5)	3.6%	0%	0%	100%
RetPD	(5,5)	100%	92.8%	95.8%	100%
RanPD	10	0%	27%	0%	37.4%
	40	0%	0%	0%	16.2%
Intermittent (10, 4, 5)	5	100%	100%	100%	100%
	50	83.8%	50.3%	98.5%	100%

* False positive rate for normal cases

Comparing against the long-term profile, the TDSAM can easily detect such deviation and raise alarm. However, it is hard for the position metric to detect an attack that generates a distribution of packet reordering similar to the long-term profile. In this case, we say that the short-term distribution is long-term-profile-like. Note that for the Heidelberg, NCU and SingNet sites, the number of total data packets transmitted in a connection is about 4000. Therefore, the dropping patterns, (20, 200, 5) and (100, 40, 5), evenly distributing the victim packets over the connection, may generate a long-term-profile-like distribution. Consequently, their Q values could be too small to raise alarm. From Table 2, we see that the false negative rate (miss rate) of the TDSAM for such kind of attacks can be as high as 100%. For the NCU site, the high

detection rate found for the PerPD attack pattern (20, 200, 5) is because the normal distribution of packet reordering is not uniform. For the UIUC site, the number of data packets transmitted was high (11300). The two attack patterns, (20,200, 5) and (100, 40, 5), do not generate an even distribution in this case. Instead, all victim packets fell into the first 2 bins. Therefore, the corresponding detection rates are also very high.

The results also demonstrate that the TDSAM performs poorly at detecting random packet dropping attacks. Such attacks generate a nearly even distribution of packet reordering in aggregate. Since the distribution is long-term-profile-like as well, TDSAM using the position metric can be easily fooled. It appears that increasing the number of bins does not help much. We observe a non-zero detection rate for randomly dropping 10 packets at the NCU site. It is also due to a non-uniform distribution of packet reordering during the normal (non-attacked) short-term period.

TDSAM has a comparatively high detection rate for retransmission dropping attacks. Although one attack may impact the distribution of packet reordering little, the aggregated impacts of many attacks can result in abnormally high occurrences of packet reordering at a specific position.

For intermittent attacks, the performance of TDSAM depends on the attack interval, binning mechanisms and specific sites. Generally, attacks with a small attack interval are easily detected, and an increasing *nbin* correlates with an increase in the detection rate. For the sites experiencing a low packet reordering rate, such as SingNet and UIUC, the detection rate remains high when the attack interval becomes large.

From Table 2, we note that *nbin* is not highly related to detection rate. For instance, for the Heidelberg and NCU sites, the highest detection rate for PerPD (20, 200, 5) is found at *nbin*=10, while the highest detection rate for intermittent attacks occurs at *nbin*=20. Similarly, the false positive rate is not highly related to the number of bins either. When increasing the *nbin*, we observe an increase in the false positive rate at the Heidelberg and NCU sites, a decrease at the SingNet site and a fluctuation at the UIUC site. Although a large *nbin* usually corresponds to large *Q* value due to the finer representation of long-term behavior, it does not mean a higher detection rate or false positive rate. This is because the *Q* test uses the tail distribution of the *Q* value rather than the *Q* value itself for intrusion detection.

We conclude by noting that we are measuring performance only at the receiver side; no cooperation is required by the sender, or from the network. For this reason, we believe our method is quite general. Our results demonstrate that detection of dropping attacks by host systems attached to the network can be very successful. We are also investigating whether interference by other hosts can cause problems similar to those caused by compromised routers [19], and if so, whether our method can likewise detect such an occurrence.

4. Conclusions and future work

As mentioned, ARQoS is a large project addressing prevention and detection of QoS attacks at the connection and packet levels. Other aspects of ARQoS, not described here due to space limitations, include:

- Selective authentication of QoS signaling, particularly of RSVP messages, to detect forgery or illegal modification.
- Pricing of DiffServ flows.
- Application of pricing to TCP congestion control, and to server access.
- Consistency checking between security policies and security mechanisms.
- Distributed monitoring of DiffServ behavior, and detection of attacks on DiffServ.

We believe the ideas and techniques presented will be valuable for protecting emerging QoS capabilities from misuse or attack. Without such security mechanisms, we predict that the benefits of such capabilities will be seriously degraded.

Acknowledgments This work has been supported by the Defense Advanced Research Projects Agency under the Fault Tolerant Networks Program, managed by AFOSR under contract F30602-99-1-0540.

5. References

[1] C. Aras et al., Real-Time Communication in Packet-Switched Networks, *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994, pp. 122—139.

[2] L. Zhang et al., RSVP: A New Resource Reservation Protocol, *IEEE Network Magazine*, Vol. 9, No. 5, September 1993.

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

[4] L. Walras, *Elements of Pure Economics* (translated by W. Jaffe), Richard D. Irwin Publ. Co., 1954.

[5] E. Fulp, *Resource Allocation and Pricing for QoS Management in Computer Networks*, Ph.D. thesis, Department of Electrical and Computer Engineering, N.C. State University, August 1999.

[6] H. Varian, *Microeconomic Analysis, 3rd ed.*, W.W. Norton and Company, Inc., 1992.

[7] N. Anerousis and A. Lazar, A Framework for Pricing Virtual Circuit and Virtual Path Services in ATM Networks, *Proc. of ITC-15*, 1997, pp. 791-802.

[8] F. Kelly et al., Rate Control for Communication Networks: Shadow Prices, Proportional Fairness, and Stability, *J. of the Operational Research Society*, Vol. 49, 1998, pp. 237-252.

[9] C. Courcoubetis et al., Integration of Pricing and Flow Control for ABR Service in ATM Networks, Proc. of GLOBECOMM, IEEE, 1996, pp. 644-468.

[10] D. Ferguson et al., Economic Models for Allocating Resources in Computer Systems, in *Market-Based Control of Distributed Systems*, ed. S. Clearwater, World Scientific Press, 1996.

[11] J. Murphy et al., Distributed Pricing for ATM Networks, *Proc. of ITC-14*, 1994, pp. 1053-1063.

[12] A. Kolarov and G. Ramamurthy, Comparison of Congestion Control Schemes for ABR Service in ATM Local Area Networks, in *Proceedings Of GLOBECOMM*, IEEE, 1994, pp. 913-918.

[13] J. Boyle et al., The COPS (Common Open Policy Service) Protocol, IETF RFC 2748, January 2000.

[14] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[15] Van Jacobson, Congestion Avoidance and Control, *Proceedings of SIGCOMM'88*, August 1988.

[16] SRI, The NIDES Statistical Analysis System for Intrusion Detection, at http://www.sdl.sri.com/nides/.

[17] Jay L. Devore, *Probability and Statistics for Engineering and the Science*, Brooks/Cole Pub. Co., 1991.

[18] Vern Paxson, End-to-End Internet Packet Dynamics, *Proceedings of SIGCOMM'97*.

[19] S. Savage et al., TCP Congestion Control with a Misbehaving Receiver, *ACM Computer Communications Review*, October 1999.