Dynamic Authentication for High-Performance Networked Applications

This work was funded by DARPA, Aasert grant DAAH04-96-1-0209. Patent pending.

Phyllis A. Schneck and Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
phyllis,schwan@cc.gatech.edu

February 6, 1998

Abstract

Both government and business are increasingly interested in addressing the growing threats imposed by the lack of adequate information security. Consistent with these efforts, our work focuses on the integrity and protection of information exchanged in high-performance networked computing applications such as video teleconferencing and other streamed interactive data exchanges. For these applications, security procedures are often omitted in the interest of performance. Since this may not be acceptable when using public communications media, our research makes explicit and then utilizes the inherent tradeoffs in realizing performance vs. security in communications. In this paper, we expand the notion of QoS to include the level of security that can be offered within performance and CPU resource availability constraints. To address performance and security tradeoffs in asymmetric and dynamic client-server environments, we developed Authenticast, a dynamically configurable, user-level communications protocol, offering variable levels of security throughout execution. The Authenticast protocol comprises a suite of heuristics to realize dynamic security levels, as well as heuristics that decide when and how to apply dynamic security.

To demonstrate this protocol, we have implemented a prototype of a high performance privacy system. We have developed and experimented with a novel security control abstraction with which tradeoffs in security vs. performance may be made explicit and then utilized with dynamic client-server asymmetries. This abstraction is called a security thermostat [12], and interacts directly with Authenticast to enable adaptive security processing. Our results demonstrate overall increased scalability and improved performance when adaptive security is applied to the client-server platform with varying numbers of clients and varying resource availabilities at clients.

1 Introduction

Both government and business are increasingly interested in addressing the growing threats imposed by the lack of adequate information security. Consistent with these efforts, our work focuses on the integrity and protection of information exchanged in high-performance networked computing applications such as video teleconferencing and other streamed interactive data exchanges. For these applications, security procedures are often omitted in the interest of performance. Since this may not be acceptable when using public communications media, our research makes explicit and then utilizes the inherent tradeoffs in realizing performance vs. security in communications. In this paper, we expand the notion of QoS to include the level of security that can be offered within performance and CPU resource availability constraints.

Performance/security tradeoffs in communications are exacerbated by asymmetries in host vs. client resource availability. For example, consider a client-server streamed application with multiple

connections from a server to clients that differ both in capability and in their current availability of resources. Moreover, there may be differences required by individuals across connections. This results in several problems. First, to maintain appropriate levels of communication performance and security, the server must be cognizant not only of client capabilities, but also of dynamic resource availabilities at those clients. Second, as security needs change, dynamic adjustments must be made across both server and clients for high performance across multiple secure connections being maintained. Finally, resource availabilities also vary as new connections are added and older connections terminate.

To address performance and security tradeoffs in asymmetric and dynamic client-server environments, we developed Authenticast, a dynamically configurable, user-level communications protocol, offering variable levels of security throughout execution. The Authenticast protocol comprises a suite of heuristics to realize dynamic security levels, as well as heuristics that decide when and how to apply dynamic security. Authenticast tracks security levels specified by each connection, and treats this as a user-level system resource component of a user's requested QoS.

To demonstrate this protocol, we have implemented a prototype of a high performance privacy system [14] with strong public key security which enables experimentation with strongly-authenticated video transmission. Using this prototype, we have developed and experimented with a novel security control abstraction with which tradeoffs in security vs. performance may be made explicit and then utilized with dynamic client-server asymmetries. This abstraction is called a *security thermostat* [12], and interacts directly with Authenticast. In transmissions which require security operations at the clients, we show performance improvements when using the security thermostat with multimedia streams. For these transmissions, we develop heuristics that enable the thermostat to do selective, per-packet authentication, accompanied by mechanisms that provide end users with feedback to control the level of security of the data they are currently receiving. Security level is defined as the percentage of data that are authenticated – thus the percentage of data whose origin is verified before those data are processed by the client. Heuristics vary security levels to remain within a user-specified range, while adapting to changing CPU resource availabilities.

In summary, Authenticast, with its security thermostat concept and dynamic authentication heuristics results in adaptive security processing. The applications benefiting from adaptive security are those in which it is possible to authenticate a user-specified percentage of the data being processed, without rendering the application unusable due to performance degradation. For example, an interactive video conference may be MPEG-encoded and distributed to interested parties. Some parts of the conference may contain sensitive issues. Those frames may be *signed*, indicating to receivers that they may want to *verify* the signed frames before playout for proof of origination. Dynamic security levels enable selective authentication. Furthermore, the same video conference may contain time-crucial information, where it is more important to receive some partially-authenticated data immediately, even though the security level is lower. Adaptive security offers the ability to make such dynamic security and performance tradeoffs.

A second set of applications for which adaptive levels of security may be of use are distributed air traffic control systems. Here, streams of data are sent to various clients, each potentially requiring different levels of authentication, depending on the sensitivity of the data being transmitted over a particular connection. At the same time, performance in terms of timely data distribution is crucial to the correct usage of these time-critical distributed applications.

The heuristics presented herein provide users of such applications the option to lower the priority of security to enhance performance, with the stipulation that security will be provided whenever it is not harmful to performance. Both services may be provided when client resources are plentiful. During times of increased load, security can be decreased, within user specification, in favor of more timely communications.

Current trends in electronic commerce are leading toward the use of security level categories to identify groups of connections or even individual transactions that require certain amounts of security. This creates another context for the use of dynamic security levels. For example, for a transaction to succeed, a receiver/client may demand a greater level of security than what is originally offered by the sender, thus forcing the sender to sign a greater number of packets so that they may be verified upon receipt. Increased signing consumes resources, and thus performance implications for that particular transaction or for other transactions involving the same sender. Our heuristics address these implications by offering options to 1) dynamically adjust CPU resource allocations as needed, and 2) remain within a user-specified security range even in situations of lesser resource availability.

We note that the purpose of our work is not to provide standards to identify and label security levels or categories. Instead, we aim to expand the notion of QoS to include system resource availability for, as well as to demonstrate the feasibility of, adaptive

security. The opportunities and tradeoffs presented by adaptive security methods are demonstrated with a common distributed application, streamed authenticated data.

Our work comprises the following contributions:

- We develop the dynamically configurable Authenticast communications protocol offering variable levels of security throughout execution within a given range, permitting modifications if necessary. Authenticast comprises the dynamic authentication heuristics presented herein, and it is responsible for the non-user-driven decisions for changes in heuristics in response to CPU resource availability changes. The Authenticast protocol also serves as CPU resource manager, in comparison to the the DRRM [13], which manages network resources.

- We improve platform scalability and performance of secure communications in a heterogeneous client-server environment. This is achieved by developing the concept of the *Security Thermostat* [12] which enables dynamic runtime modification of `securityLevel` for each connection by communicating with Authenticast. The thermostat setting is mapped to a user-level QoS specification expressing a range of permissible security levels.

- We provide feedback to end users depicting the security level of the data presented to them.

  Our results, detailed in Section 5, demonstrate overall increased scalability and improved performance when adaptive security is applied to the client-server platform with varying numbers of clients and varying resource availabilities at clients.

The remainder of this paper is as follows:

Section 2 presents some work related to our research. Section 3 discusses the flexible authentication capabilities that our heuristics provide for end users. Section 4 presents the overall architecture of the Authenticast protocol, the rationale for some decisions we make on implementation and metrics, and detailed descriptions of the dynamic authentication heuristics that we offer. We present our experimentation results and discuss some lessons learned in Section 5, and our conclusions and future work directions are in Section 6.

2 Related Work

Numerous systems, protocols, and user applications are currently being enhanced to provide greater security and thus increased flexibility for systems and end-users.

[12] details the concept of the security thermostat as well as how this is coupled with the Authenticast dynamic authentication protocol. This paper explores the topic of adaptive security, by development of and experimentation with runtime heuristics that modify the security thermostat in response in response to changes in client behavior as needed.

In comparison to varied levels of security, a large body of work has been done addressing the categorization of other application behaviors into "levels". For example, Huard, Inoue, Lazar and Yamanaka address traffic classes in [5], where network services are mapped to traffic classes as defined by the COMET group. There exists substantial indication that future trends in network security may lead to "security classes," creating the need for similar mappings in the security context, such as those presented in our work in this paper. [5] also addresses adaptive mechanisms by which a host may dynamically adjust its parameters to prevent QoS violation. Our work addresses adjustments that also encompass security as part of overall QoS. We provide user-level heuristics to best use the allocation of resources provided in a resource reservation protocol, as defined in [2]. However, we also leave room for renegotiation as defined in [13].

Nahrstedt and Steinmetz present further schemes for dynamic resource allocation in networked multimedia systems in [11]. Our work, although it addresses CPU resources rather than network resources, is similar in spirit to that presented in [11]. We are expanding on this dynamic adaptation trend to provide for security processing.

SECMPEG [10], has some parallels with Authenticast in that it offers varied levels of security for encrypted MPEG. SECMPEG includes the capability to encrypt only the most important and significant data, in order to improve performance. In comparison, we are addressing not only when and how to apply security but also how to deal with asymmetries in multiple clients receiving secure media streams. In addition, the SECMPEG security levels are based on the types of MPEG frames encrypted. Although we are currently investigating selective authentication based on frame type, we emphasize the provision of dynamic user and application-driven varied security levels for any type of application that could desire modified security levels during application execution.

Varied levels of security are also employed in the MPEG player described by Campbell et al in [8], which also considers the issue of security vs. performance, yet focuses on encryption whereas we emphasize authentication. As with [10], [8] also employs

information, such as frame type, to select the particular frames to be encrypted. Extensions of our work with authentication could use similar tactics for dynamically identifying frames to be verified.

3 Enabling Flexible Authentication

Security requirements vary with information sensitivity and user desires. The ability to comply with user-specified requirements varies with available host CPU resources, and, in some cases, network resources. By providing a flexible authentication capability, we can more closely tailor authentication provision to these categories. Tailoring combined with variable security adaptation allows us to address cases where security functionality may be sacrificed in exchange for improved performance.

Three main functions provided by strong public key authentication[1] are 1) proof of user identity, 2) data integrity, and 3) non-repudiation, which is positive proof that a given set of data were sent by a given user, and that a user cannot deny sending those data. User identity and proof of identity come from the verification of data by a receiving host.

When both a public and private key are used, the sender signs data with his private key, and the receiver "verifies" those data upon receipt with the public key of the sender. If the verification algorithm does not produce the result that would come from applying the correct public/private key pair, then the proof of identity and/or data integrity cannot be guaranteed for those data. Assuming that the receiver has the correct public key, then a failed verification indicates that the sender did not have the correct private key (and thus may not have been who he/she claims to be!) or that the data changed in transit. Neither case is acceptable to a receiver that desires proof of data origin or integrity. In addition, this public/private key authentication tactic can also provide non-repudiation.[2] Since we know by definition that the sender is ostensibly the only party with the sender's private key, then a sender cannot deny having sent any data that verify as having been signed with that sender's private key. Strong authentication often has significant system resource requirements, depending on the frequency of signing and verifying, the algorithms employed, and the lengths of the keys used.

Our work in flexible authentication provides the following capabilities:

- Parametric variation at two levels:
  1. Choosing a dynamic authentication heuristic

     A user may select from a suite of dynamic authentication heuristics detailed in Section 4.3, providing a full range of authentication frequencies as well as performance improvements.
  2. Parameterizing heuristic behavior

     The user may select security QoS ranges (such as security level), require that one heuristic take precedence over another, or leave these tasks to the automated option provided.
- Choosing an authentication algorithm and parameters (e.g., key and key length)

  A user (or the automatic heuristic) may choose to employ any of these above authentication tactics using different signing and verification computation algorithms, such as RSA [9] and DSA. [9] These algorithms differ in their on-line signing and verification processing overheads; RSA is sender-heavy, whereas DSA imposes a higher load on the receiver.

In this paper, we explore conditions in which the use of certain algorithms is warranted, and we characterize those environments in order to better tailor authentication services to performance requirements. We demonstrate how certain heuristics that dynamically modify security levels or change security algorithms can allow applications to execute with a specified degree of strong authentication while avoiding many of the performance bottlenecks often associated with these type of security procedures. Experimental results in Section 5 illustrate specific tradeoffs with flexibility, overall security level, and the performance implications of both.

[1] We note that data integrity and authentication may be provided at much less computational cost with only message digests [9] and no signature or verification overheads. However, this will not satisfy those users whose transactions require strong public key authentication, or a method that can fully guarantee proof of origination (message digest functions cannot always offer this guarantee due to collisions in hash functions). Often, policies dictate strong security, even if CPUs cannot comply. Therefore, we use strong public key authentication to demonstrate and highlight the tradeoffs that come with the highest levels of data security.

[2] We do not provide the adjudication process or third-party judge [9] to enable the non-repudiation service; we employ public key algorithms so that the mechanisms are in place should a user decide to support non-repudiation.

4 Implementation

4.1 *Authenticast*: User-Level Dynamic Authentication Protocol

4.1.1 Architecture

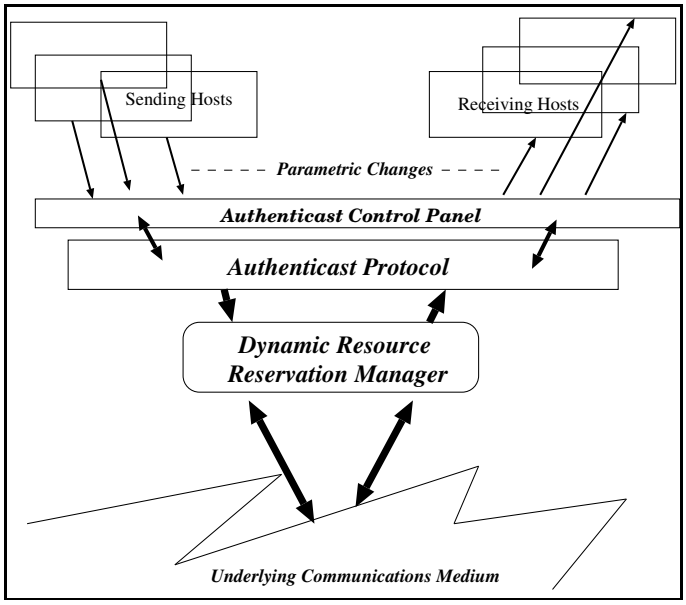

Figure 1: Position of Authenticast protocol in overall implementation architecture

Users interact with Authenticast by issuing commands and viewing feedback via the security thermostat (this expansion of the thermostat concept can be considered an "Authenticast Control Panel"), shown in Figure 2. These interactions are where parameter changes (e.g. security level or heuristic) are issued. For instance, if the user chooses to allow the protocol to manage all adjustments given a specified range, then the user can enter that range into the thermostat as well.

Authenticast and its applications communicate with the underlying network medium through the Dynamic Resource Reservation Manager [13]. The DRRM provides the capability to renegotiate a new connection without interrupting application execution. This is used to implement the **Secret Key Connection** (also referred to as "Secure Connection") heuristic, in which authentication is used in connection negotiation. The DRRM provides a secure connection and facilitates the connection change with no visible interruption in service to the application or user. A Secret Key Connection is only created at a user's request.

4.1.2 Why software?

Our dynamic authentication heuristics are implemented in software, as are the algorithms (DSA and RSA) that we employ to create and verify digital signatures. We chose software implementation over hardware (e.g. algorithms on "smart cards") for the following reasons:

- Flexibility
  One of the options we offer in dynamic authentication is an on-line change in algorithm, key, or algorithm parameters. This cannot be done with hardware. Furthermore, a software implementation allows users to employ their own (e.g., company proprietary) security algorithms which may not be available in hardware.

- Performance
  CPU speeds are increasing faster than memory speed. This implies that hardware-based security coprocessors will experience increasing overloads due to memory performance. Non-CPU processing could create potential bottlenecks and may also cause unacceptable reductions in CPU processing speeds due to caching effects.

4.2 Performance Metrics

We measure performance based on information retention, which we quantify as the *percentage of packets that are received, verified, and successfully processed.* When a client cannot compute verification algorithms fast enough to take in all of the data arriving from the network, we experience information loss. It is this loss that affects overall application performance.

We measure security-based CPU load (on the server or on one of the clients) as the *number of security operations performed over time.* Consider the following scenario: A client is receiving and processing five different data streams, where only one stream requires the verification of all packets received and the other streams require the verification of one out of every ten packets received. We use the rate at which packets are received from each stream and the percentage of packets to be verified to derive a value for the number of security operations (verifications in this case) per second metric. The security-based load on the CPU at that client may end up being far less than that at another client receiving only three streams, where each of those three streams demands that every packet be verified. Since these security operations are highly computation-intensive, the security-based load has a far greater effect on the

overall CPU load than sending, receiving, or even playing out data.

Experimental results not reported in this paper validate the effect of CPU overload on multimedia transmission. For example, when our ATM client-server connections have been granted the necessary bandwidth reservation, non-secure transmissions imposed no user-recognizable additional load due to CPU overhead. As security operations were added, we selected particular hosts that became unable to maintain packet processing rates when also faced with the added computational load of signing and verifying operations.

4.3 Heuristics

This section provides descriptions of the various heuristics used to enable dynamic authentication. Table 1 provides a comparison of these as well. We do not include the Algorithm Change heuristic in this table, because all of the heuristics herein can support either the DSA or the RSA algorithm, and, as we discuss later, a user may choose not to permit algorithm changes at all.

1. **Percentage-based Authentication**

   We define *percentage-based verification* as verifying a certain percentage of packets before those data can be processed. This percentage is dynamic, and can be modified by the user through the GUI illustrated in Figure 2, or automatically changed by an adaptive heuristic that modifies this value with changes in system resource availability or user requirements. Percentage-based verification directly trades performance vs. security level.

2. **Delayed Authentication**

   This provides an option to decrease the verification frequency of received information without reducing the overall level of authentication that is performed. Thus, every incoming packet, or whatever fraction the user specifies, will be verified before playout. Received information is accumulated in a buffer (size chosen by the user), and when that buffer fills, all of the received information is authenticated in one verification operation. The verification of this larger buffer takes no more time than the verification of a single received packet. This is because all data to be verified are hashed [9], which compresses them to a standard size, independent of how much data are hashed. For example, the product of hashing 90 packets' worth of data is the same size as the result of hashing a single packet's data. The disadvantage of this method is that the receiver may choose not to process data as they come in, but, instead wait until they are all verified. Further, if one data packet does not verify, then the whole buffer must be discounted and possibly re-transmitted. The advantage is that every received datum is verified at a significantly-reduced processing cost.

   There are two inherent tradeoffs in the Delayed Authentication heuristic. First, since a large number of packets may be verified at once, if just one of those packets was incorrect, then the entire group may not be processed. Thus, performance may be lost in that a loss of a large amount of streamed data causes an interruption potentially visible and annoying to a user. This scenario is characterized as "loss risk" in Table 1. The second disadvantage to Delayed Authentication is end-to-end delay. This delay, the time spent for a given packet from transmission at the sender to playout at the receiver, increases for each packet because that packet waits for all of the other packets in its authentication group to arrive before it can finally be verified and processed. However, in neither case does the user sacrifice any security.

3. **Secret Key Connection**

   This heuristic, invoked only at user request, provides the option to use a "secret key" [9] shared between a given sender and receiver, instead of a public and private key for each. This secret key, known only by that sender and receiver pair, can be used to scramble data. The security algorithm processing overhead with this approach is significantly reduced from that of the DSA and RSA signing and verification procedures, yet the use of a shared secret key does not provide non-repudiation, explained in Section 3. Since the key is known by *both* the sender and receiver, the sender can actually deny that certain data were sent and may choose to blame the receiver. If users do not require non-repudiation, then they may choose to enhance their performance by selecting the secret key authentication method over public/private key pairs. In fact, Authenticast can orchestrate this change using the DRRM [13], if agreed upon by both hosts, during application execution, and it will indicate to both hosts when the change occurs. The process of obtaining this new connection is discussed in Section 4.1.1.

4. **Algorithm change**

   Certain authentication algorithms such as DSA are more computation-intensive in *verification*, which is done at the client. Further, DSA can actually be extremely efficient in the signing, because signature generation requires a single

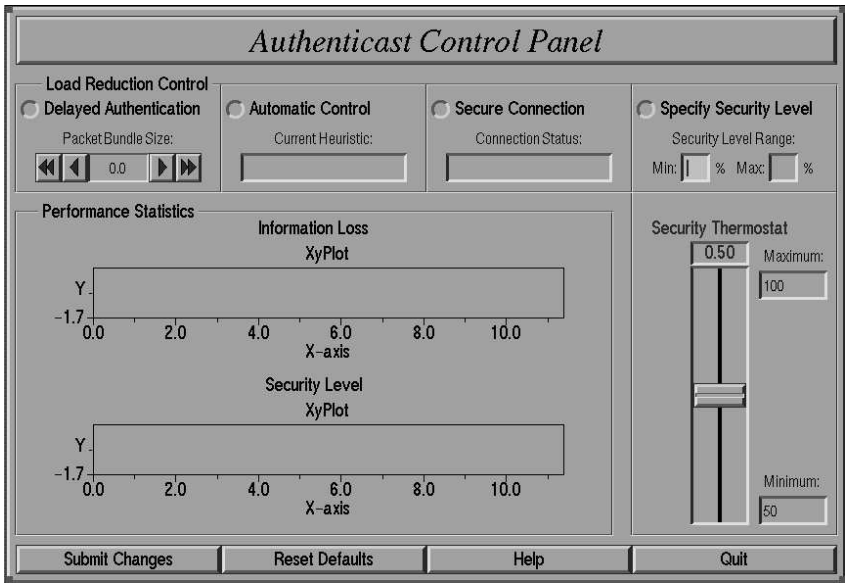

Figure 2: Expanded Security Thermostat: Authenticast Control Panel

modular exponentiation [9], which can be precomputed and, thus, not consume CPU cycles during application execution. Other algorithms, such as RSA, have the opposite load and are more computation-intensive in *signing*, done by the server.

The algorithm change heuristic exploits the fact that DSA and RSA have these respective "heavy" CPU requirements at opposite ends. In situations of extremely scarce resource availability at one end of a connection, performance can be enhanced by using a heuristic that is less of a burden to the already overloaded host. For example, if a stream is currently being signed and verified using DSA, and the client does not have adequate CPU cycles to complete the required verifications, our heuristics allow RSA to be used to sign and verify that stream instead of DSA. In effect, this shifts the heavy security burden for that stream from the client to the server.

When an algorithm change is signaled (either by the user or by a trigger in the heuristics), the sender immediately begins signing packets using the new algorithm and sets the *algorithm* flag in the packet header to match the new algorithm. Upon receiving packets, the client always looks at the algorithm flag and verifies the incoming data based on that flag. Thus, in the case of an algorithm switch, the receiver would simply verify using the new algorithm as soon as it started receiving packets with the *algorithm* flag reflecting the change. A key point to note here is that there is no interruption in signing or verifying, and thus no heuristic-based decrease in security. Further, a user always has the option to specify "no algorithm change."

We emphasize that we use the Algorithm Change option sparingly. There are frequently policies in place for certain applications that require a specific algorithm, therefore not permitting such a change. It is important to note that the other heuristics presented are independent of the algorithm being used, and exploit characteristics of the operations being performed.

4.4 Benefits comparison of Dynamic Authentication Heuristics

Table 1 compares the advantages and disadvantages to using byPercentage Authentication, Delayed Authentication, or establishing a Secret Key Connection. "Loss Risk" refers to the probability not being able to process an entire "bundle" of packets that were signed together, as a result of a failed verify operation. In this case, even if only one packet is in error, the whole bundle is assumed unable to be authenticated and those data must be retransmitted to be used. Such instances can great interruption in playout equal or greater than that related to security processing. A higher bundle size clearly means that more data is at risk for being unusable due to an unsuccessful verification. Delayed Authentication is the only heuristic presented that poses this problem, yet it also has the advantage of offering the 100% security level with very few security operations required. "Enable QoS" refers to the level of QoS the

Heuristic	Non-Repudiation	Loss Risk	Enable QoS	SecurityLevel
byPercentage	YES	LOW	varies with %	0 - 100%
Delayed Authentication	YES	varies with bundle size	YES	100%
Secret Key Connection	NO	LOW	YES	100%

Table 1: Comparison of benefits of dynamic authentication heuristics

heuristic can generally offer to the user.

4.5 Adaptive Authentication: Protocol-Driven Combination of *Percentage-based Authentication*, *Delayed Authentication*, and *Algorithm Change*

The *Adaptive Authentication* heuristic is designed to employ a combination of percentage-based and delayed authentication based on current system resource availability. The Authenticast protocol makes the modifications, triggered by changes in application behavior or system load. All modifications are made within user-specified ranges. For example, security level may be decreased during periods of high system load, but that level of authentication will never drop below some user-specified minimum.

In many cases, when security operations are decreased, the security level decrease is temporary, and, when the client stops losing packets due to CPU overload, delayed authentication may be reinstated with a larger group size, thereby restoring the security level to its state before the overload.

In this adaptive authentication heuristic, a stream may start out at full per-packet verification, using delayed authentication and signing and verifying packets in groups. If performance degrades, then the receiver senses gaps in sequence numbers of packets successfully verified and processed. In response, security level is decreased. There are two types of "gaps": The first type is that the receiver is processing a packet with a sequence number[3] less than the sequence number of the most recent packet sent by the server. The second type is that the receiver senses that the sequence number of the packet it is currently processing is greater than one more than the sequence number of the very last packet it processed. Authenticast tracks this information and will note either case as a potential cause for action. Upon sensing a gap, the heuristic switches from delayed authentication to percentage-based authentication, clearly with a percentage of less than 100% but greater than any user-specified minimum percentage.

When the server notes a server-client performance disparity (one or both sides cannot maintain performance with the current security level request, and this is signaled when DELTA exceeds LIMIT[4]), the first remedy applied is generally to switch to Delayed Authentication with a packet group size greater than 10 (if Delayed Authentication is already being used, then the heuristic would increase the size of the group of packets that are signed and verified as a unit). This allows the system to provide 100% verification, thus not decreasing whatever value the user had specified, yet it also requires far fewer signing and verification operations than even the lowest level of byPercentage (10% authentication). In order to recover from a performance imbalance such as one signified by (DELTA > LIMIT), we must allow the lagging host(s) to catch up, and thus sharply decrease the amount of security operations required.

Once the sender and receiver stabilize, we can go to a greater frequency of verification by either decreasing packet group size for Delayed Authentication, or by switching to the byPercentage heuristic for greater than 10% authentication.

We do not incorporate the option of algorithm change into the heuristic algorithm flow for two reasons: 1) The heuristic shown is independent of the particular algorithm being applied and 2) a user may specify a particular algorithm must be used, and therefore we cannot rely on algorithm change as a principle remedy for performance disparity.

We do implement algorithm change in cases where a complete workload shift from one host CPU to another would benefit both the satisfaction of user security requests for a given stream as well as performance of the overall client-server system, including other streams.

Dynamic adaptation as described above does not

[3] Each packet is given an "Authenticast sequence number," which is part of the Authenticast control information sent with each packet [12], and is independent of any sequencing identification assigned at the network level

[4] (Note that we define *LIMIT* to be a system-specific maximum performance disparity between the server and client host. The value of LIMIT is unique to each server-client pair, as it is dependent on available resources on each host. LIMIT is a conservative value, in that performance disparities must be caught *before* they cause problems visible to the user. Experimentation has shown that these situations generally worsen with time, but can be remedied if caught early.)

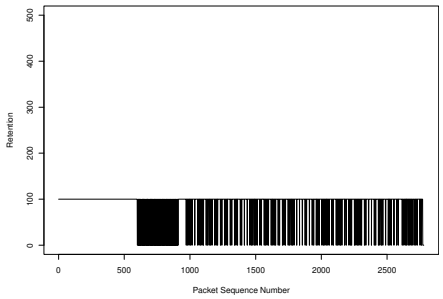

Figure 3: Example: An unacceptable Stream

create negative security implications for the user, because 1) using Delayed Authentication (even though large packet group sizes may lead to increased risk of one bad packet requiring many packets to be resent) never decreases security level, 2) the user may request that the percentage of packets that are authenticated before being processed is never decreased, and 3) a change in authentication frequency or heuristic does not create result in any type of service interruption nor does it result in security level ever decreasing below the range specified by the user at the beginning of the stream.

5 Experimental Results

5.1 Client-Server Environment Configuration

Our experimentation uses a dual 148-mhz processor Sun UltraSparc as the server. Experiment use up to seven clients, consisting of three additional dual-processor UltraSparcs identical to the server, as well as four single 167-mhz processor UltraSparcs. This hardware configuration comprises a heterogeneous client-server platform. We impose loads on various clients while they are receiving data streams. This creates dynamic changes in client resource availability.

The data transmitted are MPEG video segments, and they are streamed from the server to the client(s). Playout in each case would occur at the client as the stream is received. However, when measuring performance results, such playout processes are not performed, as this additional process would cloud the actual data processing and verification times, thereby eliminating the results by perturbating the evaluation of the Authenticast protocol heuristics.

5.2 Heuristic Performance

Figure 3 depicts the case of a single connection, where the receiver is attempting to verify data too often, and thus cannot keep up with incoming packets. The receiver's buffers fill and this results in an effective data *loss*. In Figure 3, the x-axis represents time over the entire video stream transmission, depicted by packet sequence number. The y-axis portrays whether a certain packet is retained. A value of 100 indicates that the packet is processed at the receiver, and a value of 0 indicates a loss. In this case, the receiver performs more verification operations than its CPU can handle, and thus produces an unacceptable stream. Therefore, Figure 3 illustrates a stream transmission with sufficient loss to render playout of the video infeasible.

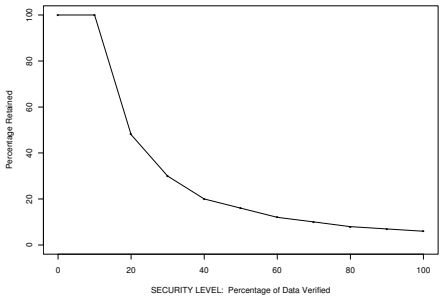

Figure 4: Variation of Information Retention Verification Percentage: 5 Clients

Figure 4 shows how information retention increases as verification frequency decreases. In this plot, the x-axis represents security level as the "percentage of data verified." The y-axis is the percentage of packets processed by the receiver. These measurements are based on data streams sent from a single server to five clients. Several trials are performed at each security level, and the results pictured represent averages. Figure 4 shows that when verification frequency is sufficiently low (under 20 percent) for this particular platform, then information retention is reasonable. However, if a user requires a greater level of security, then the "by-percentage" heuristic alone is not sufficient to satisfy security and performance requirements.

Figure 5 demonstrates two concepts. First, it depicts the effectiveness of delayed authentication as the number of clients varies. The x-axis ranges from one to five clients, and the y-axis indicates information retained. For this plot, we use a *bundle size* of 3, meaning we authenticate three packets at once. Note that this is not equivalent to using the "by-percentage" heuristic at a security level of 33 percent. Delayed authentication requires an additional signature. The sender signs each packet individually to preserve the ability of the receiver to change to the "by-percentage" heuristic at any time. However, in delayed authentication, the sender must also pro-

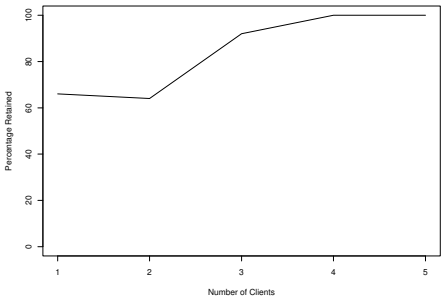

Figure 5: Variation of Information Retention in Delayed Authentication with Number of Clients

duce a signature for the entire bundle. This slows the sending rate and effectively buys the receiver more time to catch up. In fact, as the number of clients increases, so does the number of additional signatures, explaining the upward trend in retention with number of clients. Therefore, the retention in delayed authentication surpasses that of "by-percentage." This also demonstrates the second insight from Figure 5: an increased number of clients also contributes to a lower sending rate on each connection and thus improves client packet retention.

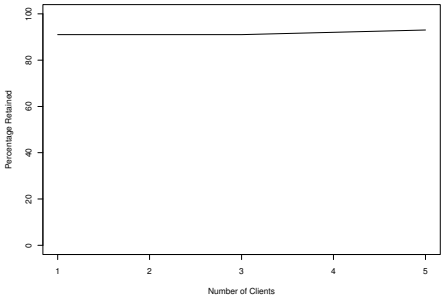

Figure 6: Variation of Information Retention in Adaptive Heuristic with Number of Clients

Figure 6 illustrates retention as it increases with number of clients when a combination of "by-percentage" and "delayed authentication" is applied. This plot again represents an average of retention in several trials for each number of clients. The transmissions start by applying delayed authentication with a bundle size of 3. The "blend" heuristic forces the security level down to below 20 percent once it senses that the receiver is lagging, as described in Section 4.3. When the receiver catches up and appears stable, the heuristic reverts back to delayed authentication (to maintain overall security level), but with a larger bundle size than before, since the receiver cannot handle the original, lower bundle size. This blend of heuristics is done automatically by Authenticast, and differs for each connection varying with system load parameters. The current security level is always visible to the user through the Authenticast Control Panel shown in Figure 2. Figure 6 illustrates the improved performance of Authenticast compared to the more simplistic approaches in Figure 4 and Figure 5.

The "diagnosis" of needing improved performance is presented in Figure 7, Figure 8, and Figure 9. These three plots are produced from the same stream as the 1-client case in Figure 6.

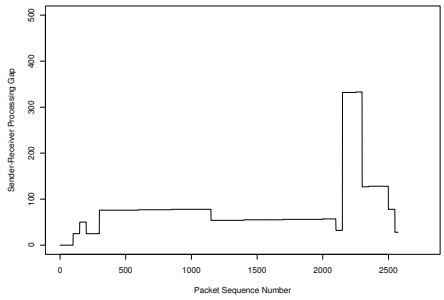

Figure 7: Variation in Sender-Receiver Processing Disparity Due to Heuristic Changes in Security Level

Figure 7 is a plot of the gap between sender progress and receiver progress, thus the sender-receiver disparity over time. Time, the x-axis in this plot, is defined by the number of packets that have been received so far at the client.

This plot, when compared to Figure 8 and Figure 9 demonstrates that higher gaps in sender-receiver progress trigger Authenticast to lower security level. Figure 7, when directly compared to Figure 8, further demonstrates that, if gaps are allowed to grow too large, this results in the client's inability to retain packets.

Figure 8 shows the data retention over the life of the stream, where the x-axis represents packet sequence number and a y-axis value of 100 indicates that the packet was retained. The information loss at this client occurs approximately between packets 2250 and 2500. Note the correlation between this loss and the increase in security level at these same packets in Figure 9. Looking at the progress gaps in Figure 7, we also demonstrate that, for this platform, a gap in sender-receiver progress of under 100 packets led to stable performance. As that gap rose, triggered by the increased security level, performance degraded.

5.3 Performance Overhead of Heuristics

Although the execution of dynamic authentication heuristics themselves does consume CPU resources, this has no affect on application performance relative to security operations. In experimentation, we

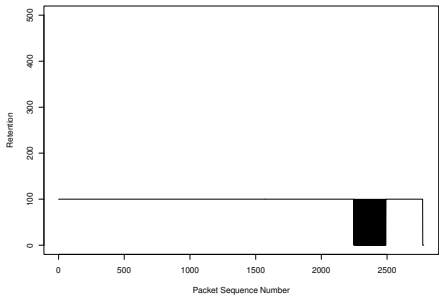

Figure 8: Greater Retention Over Time with Adaptive Heuristic

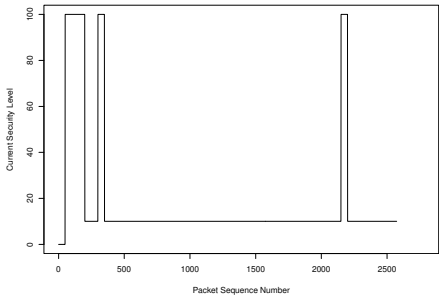

Figure 9: Security Level Variation Throughout Stream

streamed data from the server to clients and invoked all of the heuristic operations outlined in the previous pseudocode *without* actually executing any security operations. We saw no user-recognizable application performance degradation in these trials.

5.4 Lessons Learned

To mitigate performance tradeoffs with user-requested security, neither the "by-percentage" hueristic nor the "delayed authentication" heuristic is sufficient on its own. We see the sharp performance degradation with "by-percentage" in Figure 4. If a user requests 100 percent security, that will often not yield suitable application performance. Delayed authentication, with a sufficiently large bundle size will provide that 100 percent security level *and* high performance, but large bundle sizes lead to increased end-to-end per-packet playout delays.

To fulfill the overall security requests, if security level must be decreased, then we must keep that security level low for the shortest possible time. We have discovered through experimentation with blending "by-percentage" and "delayed authentication" that, for this platform, a drastic reduction in security level as soon as sender-receiver progress gaps are detected is significantly more effective than more frequent, conservative decreases. Frequently, these large decrease gives that receiver the chance to "catch up," while the sender-receiver progress gap is not yet irreversibly large. Although the blended heuristic does sacrifice security level at certain points in the stream, it does minimizes "exposure time" to a user-specified level, and produces a considerable improvement in performance over the "by-percentage" heuristic alone.

All of the experiments herein use the DSA algorithm for authentication, which, as mentioned earlier, is a greater burden to the receiver than to the sender. We are currently expanding our platform to demonstrate scalability on many more clients. In the final version of this work, we will also demonstrate results based on both DSA and RSA with a significantly larger number of clients, likely also demonstrating improvements in scalability due to bottlenecks.

6 Conclusions and Future Work

The Authenticast communication protocol provides adaptive security for interprocess communications. When using it with a client-server application, varying numbers of clients, and varying resource availabilities at clients, improvements in scalability and application performance are attained compared to non-adaptive approaches. Flexibility concerning security is attained using its Security Thermostat abstraction. Such flexibility is shown useful for both servers and clients. The server can exploit it to increase the security offered when communication rates decrease, since lower communication rates permit clients to do increased security processing for each channel. Clients can exploit flexibility in security vs. performance to balance playout rates and the resulting timeliness of displayed images against the security of the data being displayed.

The explicit control of security as well as the feedback to end users concerning realized levels of security are supported by a graphical user interface and by user-accessible application interfaces. With these interfaces, users have the capability to access the Security Thermostat to specify acceptable security ranges and to modify security levels at any time during an application's execution. Typically, however, security modifications within pre-specified ranges are performed by heuristic algorithms which attempt to realize suitable tradeoffs in performance vs. security throughout an application's execution. The Authenticast system offers multiple such heuristics, thereby addressing the diverse needs of different underlying security methods and variations in end user needs.

Our future work on adaptive authentication

heuristics includes defining a mapping of security level to end-results of the application. Instead of reading feedback from the control panel, we envision a method by which to integrate security state into the video playout itself for immediate visual realization of security or lack thereof in data being played out.

In addition, we are currently devising algorithms for the dynamic optimization of CPU resource allocation between the server and all of the clients. As security demands change on a per-stream basis, resource requests vary as well. Our challenge is to make the most efficient use of resources at all points, without the overhead of the optimization heuristics themselves causing further performance degradation. Another interesting topic for future research is the interaction of the security thermostat QoS metrics with other metrics, such as jitter, that are of known interest to the networking community. As noted previously, we are also expanding our platform to demonstrate the scalability of our flexible authentication heuristics to a much wider environment, such as a campus backbone.

References

[1] I. Agi and L. Gong. An empirical study of secure mpeg video transmission. In *Symposium on Network and Distributed Systems Security*. IEEE, 1996.

[2] A. Campbell, C. Aurrecoechea, and L. Hauw. A review of qos architectures. In *Proceedings of 4th IFIP International Workshop on Quality of Service, IWQS'96*, March 1996.

[3] W.D. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), November 1976.

[4] ATM Forum. Atm user-network interface (uni) signalling specification 3.1. *PTR Prentice Hall*, 1993.

[5] J.F. Huard, I. Inoue, A. A. Lazar Lazar, and H. Yamanaka. Meeting qos guarantees by end-to-end qos monitoring and adaptation. In *Workshop on Multimedia and Collaborative Environments of the Fifth IEEE International Symposium On High Performance Distributed Computing (HPDC-5)*. IEEE, August 1996.

[6] J.F. Huard and A.A. Lazar. On qos mapping in multimedia networks. In *21th IEEE Annual International Computer Software and Application Conference (COMPSAC '97)*. IEEE, August 1997.

[7] D. Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4), April 1991.

[8] Y. Li, Z. Chen, S. Tan, and R. Campbell. Security enhanced mpeg player. *Department of Computer Science, University of Illinois at Urbana-Champaign*, 1996.

[9] A.J. Menezes, P.C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 1997.

[10] J. Meyer and F. Gadegast. Security mechanisms for multimedia-data with the example mpeg-i-project. *Project description of SECMPEG*, 1995.

[11] K. Nahrstedt and Steinmentz. R. Resource management in multimedia networked systems. In Jim Cavanagh, editor, *Handbook of Multimedia Networking*, pages 381–405. Auerbach Publications, 1995.

[12] P. Schneck and K. Schwan. Authenticast: An adaptive protocol for high-performance, secure. Technical Report GIT-CC-97-22, Network ApplicationsGeorgia Institute of Technology, Atlanta, GA 30332-0280, July 1997.

[13] P. Schneck, E.W. Zegura, and K. Schwan. Drrm: Dynamic resource reservaion manager. In *International Conference on Computer Communications and Networks*. IEEE, 1996.

[14] Bruce Schneier. Cryptography, security, and the future. *Communications of the ACM*, 40(1):138, January 1997.