

Continually Improving Stream Analysis for Network Security

Nancy J. McMillan, (Battelle), mcmillann@battelle.org

Douglas D. Mooney, (Battelle), mooneyd@battelle.org

David A. Burgoon, (Battelle), burgoon@battelle.org

Abstract

In many real-world environments, events happen at irregular intervals and measurements describing those events are recorded, e.g., network connection attempts. The flow of measurements thus generated is stream data. The pace of real world events, which is not controlled, governs the rate at which stream data flows. The real-time management and use of stream data for decision-making and/or characterization is complicated by its inherent variable flow rate. By their nature, these events require data management and data processing based on algorithms. Data management and processing take time; the amount of time governed by the complexity of the algorithms employed. Typically more complex and time-consuming algorithms are only considered when they provide superior decision-making ability or superior characterization. However, if new data cannot be handled as quickly as it is generated, real-time management and use is not occurring. Thus, there is a natural trade-off between algorithms that store and process data quickly enough to keep up with the flow of stream data and algorithms that provide a sufficiently precise decision or characterization.

Continually improving stream analysis (CISA) is a mechanism for managing the trade-off between providing sufficiently precise decisions/characterizations and keeping up with the flow of stream data during run time. The real-time stream data monitoring features that are provided by CISA are:

1. The algorithm always provides a current decision or characterization.
2. The precision/accuracy of the current decision or characterization improves when there is more processing time available relative to the rate of data flow, i.e., more processors, faster processors, and slower data flow all translate to more precise/accurate decisions or characterizations.
3. The algorithm scales automatically to optimize accuracy/precision of the current decision or characterization as a function of data flow rate.

In this work, concepts for CISA are realized in the framework of a cyber-security example. Specifically, a CISA intruder detection system (IDS) is developed, which monitors firewall data. The IDS developed is a dynamic classification/characterization tool that first identifies groups of sources by common behavior patterns then characterizes the behavior of the groups identified over time. As new intruder behaviors emerge, they are captured by the appearance of new groups or the migration in behavior patterns of existing groups.

Introduction

Modern data stream analysis requires an integration of data management and decision support algorithms. Traditional data stream analysis typically consisted of sensor data with threshold algorithms (e.g., sound an alarm when the temperature rises above a specified threshold). The analysis was simple enough that the algorithm could be embedded in the sensor device and data management was not an issue. More advanced applications, however, require algorithms that make decisions based on the readings from not only one sensor, but a network of sensors, often measuring different quantities (e.g., temperature, blood pressure, and heart rate). Such algorithms typically involve complex multivariate statistical models. The concept of data stream has expanded to include more complicated data sources than physical sensor measurements such as text, network traffic, click-stream data, audio, and video. An important part of the analysis process is extracting meaningful information or features from the data stream. A data stream analysis system must

manage multiple data sources, enable extraction of information from the stream, and execute decision support algorithms in a timely manner.

The real-time management and use of stream data for decision-making is complicated by the variable flow rate inherent in the data. The pace of real world events, which is not controlled, governs the rate at which stream data flows. Thus, stream data can both trickle and gush. Data management and processing take time; the actual amount of time governed by the complexity of the algorithms employed. More complex and time-consuming algorithms are sought when they deliver superior decision-making ability or superior characterization. However, if new data cannot be handled as quickly as it is generated, real-time management and use is not occurring. In effect, a late answer is always an incorrect answer. Thus, there is a natural trade-off between algorithms that store and process data quickly enough to keep up with the flow of stream data, and algorithms that provide a sufficiently precise decision or characterization. Current data stream systems manage this trade-off at design time and are constructed with a fixed (or anticipated maximum) data flow and algorithm precision in mind. Under increased rate of data flow these systems typically break, providing wrong or out-of-date answers, or even no answer. Systems built for maximum flow, moreover, are oftentimes underutilized and therefore a problematic solution in cost limited environments. A number of research efforts are currently underway to address managing and querying data streams. A good overview of current research directions can be found in [7]. Several approaches to data stream management are presented in [3], [4], and [5]. Work on querying data streams is presented in [2]. Algorithms tailored for data stream use are presented in [8] and [10].

Continually improving stream analysis (CISA) is an integration of data stream management and analytical algorithms so that changes in the flow rate of the data result in changes in the precision of the answer. CISA always provides a current decision or characterization. The price of an answer always being available is that the precision is variable. The precision of the current decision or characterization improves as more processing time becomes available relative to the rate of data flow. In particular, more processors, faster processors, and slower data flow all translate into more precise decisions or characterizations. The algorithms used by CISA scale automatically to maximize the accuracy of the current decision or characterization as a function of data flow rate. As the data flow increases or processing is diverted to other tasks, the algorithm will seamlessly produce less precise answers that can be generated with the current level of resources. No decisions or manipulation of algorithms by the user is necessary. If the answers produced by CISA are not sufficiently precise, all that is required of the user is to make more processing resources available.

Both architectural and algorithmic components are essential for the implementation of CISA. The architecture establishes a convenient and flexible environment for managing and prioritizing the flow of stream data through data management, data summarization, and decision-making/characterization activities. Unlike traditional algorithms that operate on an entire dataset at once, CISA algorithms perform the decision-making/characterization activity within the architecture.

In this paper, concepts for CISA are realized in the framework of a cyber-security example; specifically, a CISA intruder detection system (IDS) to monitor firewall data.

Architecture

The CISA architecture is one of asynchronously operating objects communicating with each other. The three key components of the architecture, pictured in Figure 1, are: 1) Data Source Objects, 2) Data Management Objects, and 3) Algorithm Objects. Data source objects are the data stream inputs. Conceptually each data source is an item to be classified or characterized so a complex data stream source, such as a firewall, will be broken into the component sources of interest, such as connecting source IP address. The data management objects hold and calculate summary statistics or features to be used by the algorithms. Algorithm objects are the decision support logic which updates the classification or characterization of any, and perhaps all, data source objects as new data arrive.

The data management system and algorithm components of the CISA architecture break the processing requirements of decision-making/characterization into two functional pieces. Raw data storage,

maintenance of any necessary data buffers, and calculation of summary statistics or features from the raw data together entail one function. Application of a classification/characterization algorithm based on the summary statistics is the other. Both functions are operating simultaneously. The storage and feature calculation function, which could be computationally burdensome, can be located on a separate machine than the (also computationally burdensome) algorithm function without requiring any system redesign because they communicate only via topic messages.

In the text that follows we describe the component objects in detail.

Data Source Objects. New stream data flows into the system through autonomous data source objects. The data source objects are the originators of requests for both data management actions and classification/characterization updates performed by the other two components. These objects understand their own current classification/characterization sufficiently to prioritize their own requests to the other two components of the architecture. Their understanding of themselves comes from feedback provided to them by the other components of the system. The feedback data source objects receive is essential to both prioritizations of future requests and maintenance of a cooperative relationship between the architecture components, i.e., to ensure, in an asynchronous environment, that an individual data source object does not make duplicate requests.

The source data objects manage the trade-off between storing and processing data quickly enough to keep up with the flow of stream data and providing an accurate decision or characterization. They do this through the management and prioritization of three necessary activities: 1) data storage/buffering, 2) characterization feature updating, and 3) algorithmic solution updating. The current CISA architecture manages the prioritization of processing separately for the data management and algorithm objects. Thus, there is no attempt to control the amount of time the data management component gets the processor versus the amount of time the algorithm component gets the processor. This is possible because the architecture is envisioned to consist of components that can be distributed to multiple computers on a network.

The many feature requests that source data objects can make are necessarily prioritized. Two strategies for prioritizing the individual features are the speed with which they can be calculated and the importance in the classification routine. A second level of prioritization is done to each data source object feature request based on the priority assigned to each individual data source object. This priority is assigned to each source object based on a certainty measure associated with its current classification or characterization. The idea is that source data objects whose classifications are fairly certain or whose characterizations are fairly precise would have a lower priority than do source data objects with more uncertain classification or less precise characterizations. Thus, more processing time will be spent on sources whose classification/characterization is more in question and less processing time will be spent on sources whose classification/characterization is less likely to change. Prioritizing sources is a burden placed on the algorithm object.

The source data objects tell the algorithm object when their own features have changed. This is a signal that the algorithm object needs to update its classifications/characterizations. Requests for algorithm updates happen on a feedback loop. In other words, a source data object does not send a second new set of feature values to the algorithm until a response is received for the first set. This ensures that older low priority feature updates never override more recent high priority feature updates as the algorithm object works its way through all of its queued requests. The priority associated with each request to the algorithm is that of the source data object sending the request. After the algorithm object has acted on each update request it sends messages back to those source objects whose classification/characterization have been impacted by that request.

Data Management Objects. A data management system (object) is designed to efficiently hold and calculate summary statistics based on the data provided to it by the data source objects. The data management system is able to respond to requests from multiple data source objects respecting a global prioritization metric determined by the data source objects.

The data management object must maintain a current time window of stream data for each source from which summary statistics/feature are calculated. There are two essential data storage activities required to do this. New data must be stored, and old data must be purged. The data management object only performs these activities at the request of source data objects, since the timing for these activities must be consistent with the prioritization of all data management object tasks. Storing new data is viewed as a high priority activity. Purging old data is viewed as a low priority activity. The priorities for summary statistic/feature calculation fall between these two extremes. Many requests from the same object to store new data may accumulate in the data management prioritization queue if stream data is flowing from that object very fast. However, purge requests operate on a feedback system so that a second purge request is not sent by a single source data object unless the first purge activity has been completed.

There can be any number of summary statistic/feature calculation requests. The number is governed by the nature of the stream data being characterized. For feature calculation, the source data objects act on a feedback system that requires them to get a response from the last message they sent of a particular type before they send another of that type. Thus, at any one time the data management object could have many requests to do something from each source data object, but it could not have two duplicate feature calculation requests from the same object.

Algorithm Objects. Algorithm objects are designed to *update* the classification/characterization of, possibly, *all* data source objects based on new information provided to it in a prioritized fashion by *any* individual data source object. The algorithm embedded in this object can be any decision-making/characterization algorithm, but optimal performance is obtained when the algorithm employed can compute a new solution based on its last solution together with a set of changes to the data on which its last solution was based. Thus, for optimal performance only changes to the data source object summary statistics/features are passed to the algorithm object and the solution updated rather than re-computed.

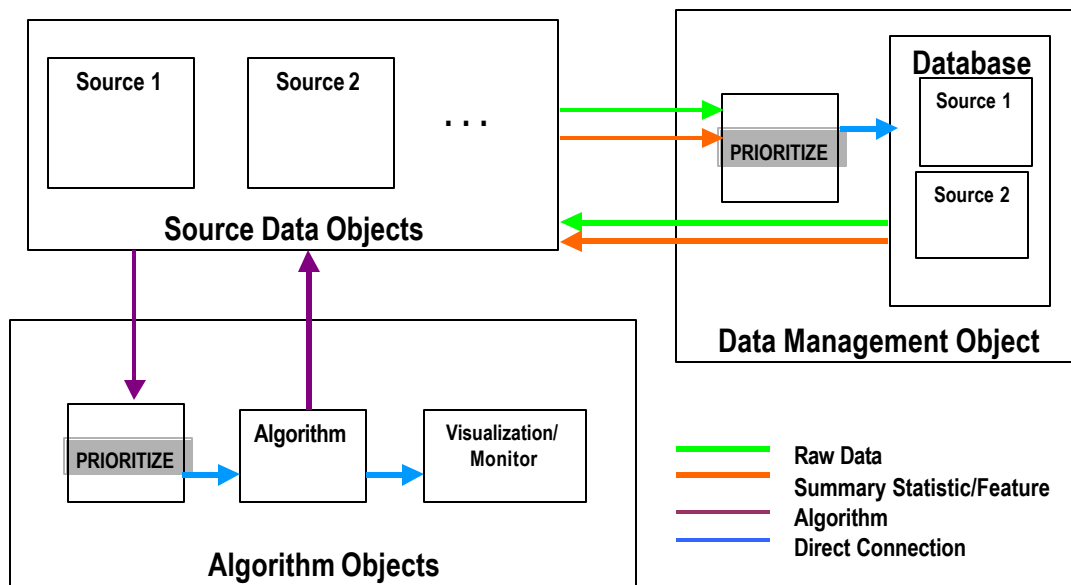


Figure 1. Three Components of the CISA Architecture

Multiple decision-making/characterization algorithms can easily be plugged into the CISA architecture either alone or simultaneously because of the functional break between the data storage/feature calculation and the classification/characterization algorithm. This is demonstrated below with two IDS algorithm examples. There is a monitoring object associated with the algorithm, which maintains a summary, either visual or numerical, of its current classification/characterization. The form of this summary is dependent on the nature of the specific algorithm embedded within the algorithm object. The monitoring object is the window through which the CISA delivers its results.

Embedded Algorithms

The traditional approach to algorithms is that a static data set is provided as input, the same algorithm rules are applied to all the data, the algorithm processes the entire data set, and finally appropriate output is generated for all cases. This approach has severe limitations in a streaming data environment and to perform well algorithms need to be restructured. Specifically:

- Algorithms must interface with the data management architecture and be able to handle assignment of priorities to features or data sources;
- In order to scale with data volume flow, algorithms may need to be broken into component pieces to provide partial answers; and
- Reduced complexity or incremental versions of algorithms may be needed.

Redesigning algorithms to have a smaller computational complexity or to be incremental in nature are their own research directions (see, for example, [6], [10], and [11]). The requirements of data streams do not necessarily require new algorithms be developed, rather it may be possible to restructure existing algorithms to accommodate streams. A wide class of algorithms can be converted to CISA algorithms. Generally any algorithm is a good CISA candidate if it exhibits some of the following properties:

1. It relies on more than one feature;
2. Some of the individual features take time to compute or measure;
3. Meaningful nested "sub-algorithms" can be built on increasing sets of features;
4. The algorithm can efficiently, incrementally update its current solution based on small changes to the feature values; and
5. The algorithm is being applied to many objects at once and there is a natural method for prioritizing objects.

Two strategies are typically used, often in combination, to convert traditional algorithms to CISA algorithms, termed **feature priority** and **data-source priority**. The key steps in each are:

- Feature Priority
 - Order features
 - Create series of nested models that use an increasing number of features
 - Develop a function to assign priorities based on feature order and current classification
- Data-source Priority
 - Order data sources
 - Assign priorities based on the uncertainty of classification or cost of misclassification
 - Incremental algorithms are usually essential

For an example of a traditional algorithm converted to a CISA using the feature priority approach, consider a statistical regression model built based on training data with a certain set of predictors. Suppose there is a meaningful ordering of predictors in terms of the time required for computation, its contribution to the model, or some combination of both. A CISA algorithm version of the regression model can be created by building a regression model on the first predictor in the ordering scheme, then one using the first two predictors, then the first three, and so on until the full model is computed. When the CISA architecture is

operating, the different features of the sources will be computed at different times depending on assigned priorities. However, for any source at any time, a predicted value with confidence bounds can always be given based on features currently computed. When data arrives quickly, models using perhaps one or two features will be used; when data arrives slowly the full model can be used, presumably giving a better answer with tighter bounds.

As a second example, consider a decision tree built on training data that has a number of splits based on multiple features. Decision trees can be pruned back to any higher level and the result is still a decision tree which can not only make classifications, but can also give probabilities for a record being in any of the classification categories. Thus, a nested set of models is immediately available, namely the pruned sub-trees. To make a CISA version of the decision tree algorithm, order the features with the first feature being the predictor making the first split, the second feature being one of the features making a second level split, and so on down the tree. This most likely will not be a unique ordering due to multiple features of a given level, but these could be uniquely ordered in terms of deviance explained or time required for feature computation. When data are arriving quickly, perhaps only the features making the first couple splits can be computed and the largest sub-tree using these few features will be used. When the data arrives more slowly, the full feature set can be computed and the whole tree can be used to assign classifications.

Cluster analysis provides an example of using the data-source priority method of algorithm conversion. To perform cluster analysis every data source needs a full feature set. The CISA conversion consists of two parts: maintaining the best current feature set in light of data flow rates and computing the clustering algorithm in a timely manner. Data prioritization is used to maintain the best current feature set. Priority assignments proceed from the following rules. Data sources with an incomplete feature set get highest priority. Typically features exhibit greater stability or experience smaller relative changes when they are based on large amounts of data. Consequently, data sources with fewer data observations receive higher priority than data sources with many observations. The more uncertain a data source's cluster assignment, the higher the priority it receives. Cluster analysis is a computationally intensive procedure so the use of incremental algorithms based on the changes in the data (see, for example, [6] and [11]), rather than re-clustering the entire dataset, are ideal for real-time clustering.

Internet Security Data

Battelle obtained enterprise-wide firewall data from an information technology company for a one-week period during the summer of 2002. The log data consisted of connection summary information for every connection attempt between pairs of computers on opposite sides of the firewall. Included in the data recorded for each connection were Date/Time, Source and Destination IP addresses, Service or Port, and Firewall Action (accept/drop/encrypt/...). Information about the packet size and contents was not available.

One use of the firewall log data is the real-time identification of external sources trying to access the company's network inappropriately, i.e., an intruder detection system (IDS). With this question in mind, the log data were conceptually organized by source IP address. Each source has some number of connections associated with it. To best address external sources accessing the internal network, the log data were filtered (using IP address ranges) to exclude connections that were originated within the company and to exclude connections whose destination was not within the company's network.

Organizing this internet security example into the CISA framework requires identification of the components in the CISA architecture. The unique sources connecting from external sites to sites within the company's network each generate a stream of data, and, hence, are the data sources. A time relevant window of the connection data for each source is the data that must be managed and summarized by the data management object. An algorithm classifying the behavior of each source into categories identifying normal behavior and inappropriate behavior must be embedded in the algorithm object. The features used by the algorithm to classify sources are summaries of the connection information belonging to each source. Figure 2 depicts the CISA architecture tailored to the IDS example.

An additional level of information is portrayed in the detailed internet security CISA architecture that was not provided in the general CISA architecture, namely the use of topics for communication between objects. The communication topics employed by the three components of the architecture are drawn in Figure 2. In Figure 2, the source data object component highlights the fact that the log data must be preprocessed into streams of data specific to unique IP addresses connecting to the company's network. The data management object communicates new features calculated back to the source data object via topics unique to each source data object. An alternative architecture would have communicated all new features calculated via a single topic to which all sources listened, similarly to how the algorithm component communicates updated classifications to the source data objects. It is unknown which scheme is superior.

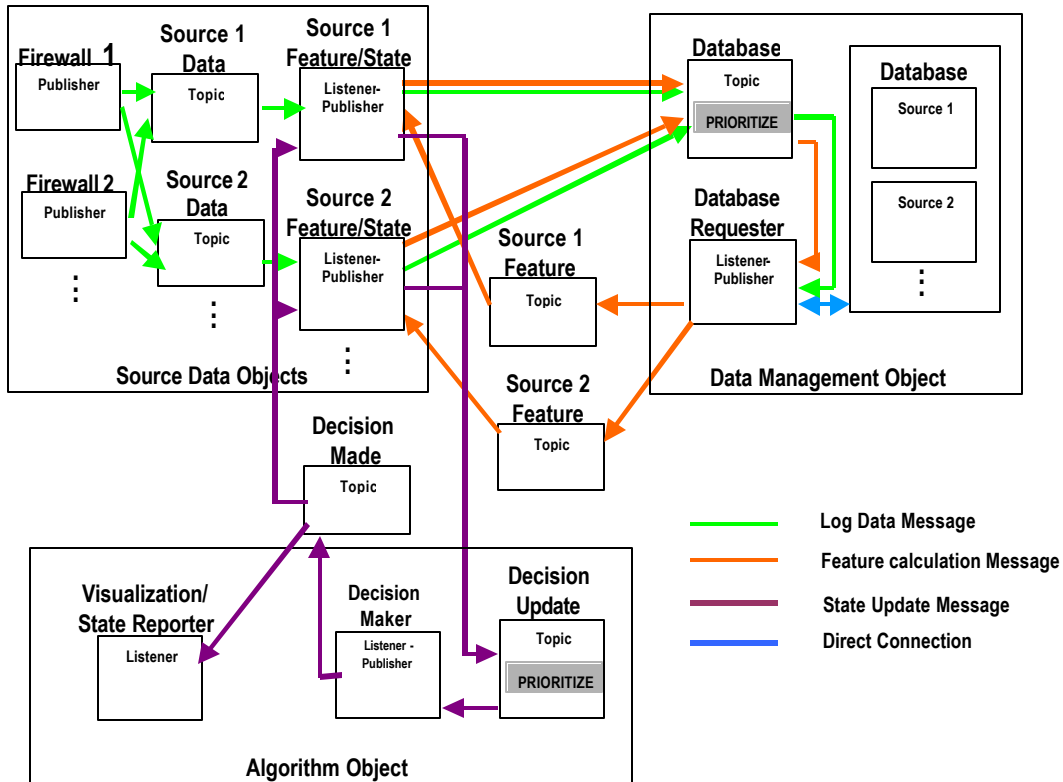


Figure 2. Internet Security CISA Architecture

Two CISA-based IDS algorithms have been developed to perform real-time monitoring of the firewall data and incorporated into the CISA IDS architecture depicted in Figure 2. The first is a feature priority **rule-based** classifier. It looks for known intruder patterns in connection data from individual sources that were identified in offline exploratory analysis of firewall data (i.e., misuse detection). The second IDS is a **dynamic** data priority classification/characterization tool that first identifies groups of sources by common behavior patterns then characterizes the behavior of the groups identified over time (i.e., anomaly detection). By dynamic we mean that as new intruder behaviors emerge, they are captured by the appearance of new groups or the migration in behavior patterns of existing groups. It does not rely on a rule set based on previously observed data. For background on IDS see [9].

Rule-based CISA

The rule-based CISA for network security was built by first performing a cluster analysis on training data, analyzing the feature set of each cluster, and then developing rules. The following feature set was used:

- Percentage of total connections dropped,
- Maximum number of connection attempts per second,

- Number of distinct internal IP addresses accessed,
- Number of distinct services accessed,
- Ratio of the number of distinct internal IP addresses accessed to the number of connection attempts, and
- Ratio of the number of distinct services accessed to the number of connection attempts.

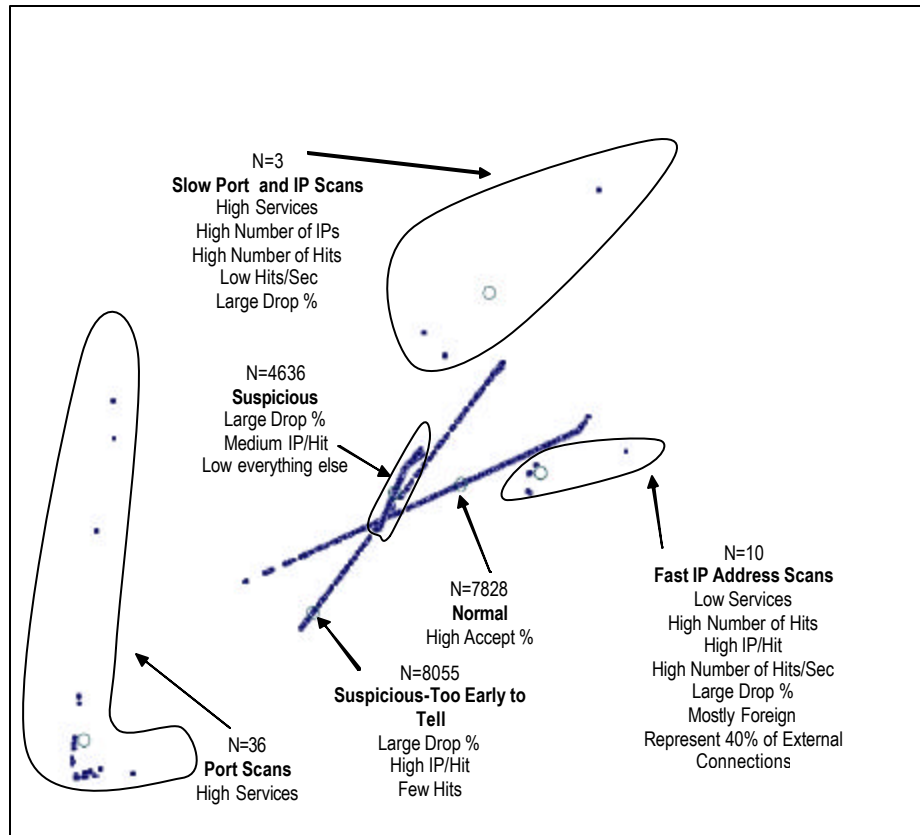


Figure 3. Cluster Analysis of the Firewall Feature Set

Figure 3 presents a visualization of the data created by first performing a principle components analysis on the cluster means (or centroids) and then projecting the data into the first two principle components. Clusters are labeled with centroid information of interest, to profile behaviors. A collection of distinct types of behavior were observed; namely, port scans, IP address scans, both port and IP scans, suspicious but too early to tell, suspicious but unknown (high drop percentage), and normal (low drop percentage). Additionally several modifiers were added to these classifications to indicate speed of connection attempts and nationality of the source (in particular foreign or domestic). Of special interest to the network security staff to whom the data belonged was the cluster of scans occurring at a very slow rate, plausibly indicative of people trying to hide their behavior within normal traffic and fairly difficult to detect.

Another way people try to hide their behavior is by performing a scan from multiple IP addresses. Either they have access to a set of addresses or “spoo” (hijack) the addresses of others (who are probably physically nearby so that results can be monitored). It is likely that the addresses in such sets are similar, differing only in their suffixes. The clusters in Figure 3 can be further grouped or clustered according to source IP addresses. We found subclusters containing between five and 20 nearly sequential IP addresses (i.e., XXX.XXX.XXX.67, XXX.XXX.XXX.68, etc.) with nearly identical feature profiles, suggesting that single individuals are responsible for them. A number of these subclusters were found within the “suspicious but unknown” cluster, each with only 2 or 3 connection attempts, all dropped by the firewall.

This cluster is characterized by having a high drop percentage, but otherwise normal profiles—likely the result of splitting the scanning activity among multiple source addresses.

Analyzing the features made several facts apparent. First, when the number of connection attempts was small, some features did not contribute to classification in a meaningful manner. Second, some features were more time consuming to compute than others. Finally, the behavior exhibited by source addresses was fairly stable over time and, in general, more data led to a more reliable feature set. For these reasons features were ordered in terms of the number of connections necessary before the feature became useful. The ordering nicely coincided with the amount of time required for feature computation. The ‘current feature set’ was taken to be the most recent value of all features ever computed. An exception occurred when newer, more highly ordered features were inconsistent with older, lower ordered features. In that case, the old features were purged.

More specifically the features “drop percentage” and “maximum number of connection attempts per second” are quick to compute and make sense with relatively few data records. Features related to port scans and IP address scans are more expensive to compute because they require creating lists of ports and IP addresses scanned for each source. The ratio measures made sense after relatively few connection attempts. The absolute counts were more reliable indicators, but became indicative of malicious behavior only after hundreds or thousands of connection attempts. For example, the company had about 65,000 IP addresses assigned to it, but only several thousand of these were in use. Sources attempting to access more address than were in use surely indicates problem behavior, but several thousand observations are required before this conclusion can be drawn.

The rule-based algorithm progresses as follows: The primary classification has five levels. Initially a source is placed in Level 0: *Too Early to Tell*. Using the “drop percentage” feature, Level 1 classifications are defined with *Normal*, *Suspicious*, and *Too Early to Tell* categories. The remaining Levels all have categories *Normal*, *IP Scan Only*, *Port Scan Only*, *Both IP and Port Scan*, *Unknown*, and *Too Early to Tell*. Level 2 uses “drop percentage” and the ratio features, Level 3 uses Level 2 features together with “the number of distinct services accessed,” and Level 4 uses all features.

The five-level classification scheme is illustrated in Table 1. A new source starts in Level 0: *Too Early to Tell*. As soon as enough data are collected for a drop percentage measure to be meaningful, the source gets a Level 1 classification. As additional data arrive and as resources are available for feature computation, the source successively moves to Level 2, 3, and 4. While these levels have the same classification categories, the addition of features in the classification process may move sources from *Too Early to Tell* or *Unknown* to another category. Sources may also move from only one type of scan to both. If, at any time, “drop percentage” indicates a move from *Normal* to *Suspicious* or vice versa, the rest of the feature set is purged and recalculated according to the established priority.

Table 1. Classifications Produced and Features Used By the Sub-Algorithms of the Rule-Based Internet Security CISA

Level	Classification					Features Added	
0	Too Early to Tell					N/A	
1	Normal	Suspicious			Too Early to Tell	Drop %	
2	Normal	IP Scan Only	Port Scan Only	Both IP and Port Scan	Unknown	Too Early to Tell	Ratio Measures
3	Normal	IP Scan Only	Port Scan Only	Both IP and Port Scan	Unknown	Too Early to Tell	Distinct Services
4	Normal	IP Scan Only	Port Scan Only	Both IP and Port Scan	Unknown	Too Early to Tell	Distinct IP Addresses

To evaluate the performance of this classification algorithm, a set of test data was chosen consisting of about 19,000 connection attempts from 108 distinct sources over a period of 2.5 hours. These data were streamed into a prototype CISA and the system was run until there were no further requests for processing. The final classification of each source was recorded and for testing purposes defined to be the ‘correct’ classification of that source. The algorithm “Level” was also recorded for each source.

To evaluate the performance of the rule-based CISA algorithm, the classification of each source after all 19,000 connections were streamed into the system was recorded at a number of data flow rates. The level sub-algorithm by which each source was classified was compared to the ‘correct’ level. Based on this comparison, each source was coded as being in one of four categories:

1. Correctly classified by same level of algorithm;
2. Correctly classified by a different level algorithm;
3. Consistently classified by a different level algorithm; and
4. Inconsistently classified.

The structure of Table 1 defines a hierarchy of source classifications. For example, *Port Scan Only* is a sub-class of *Suspicious*, which in turn is a sub-class of *Too Early to Tell*. A source is categorized as consistently classified if its classification is not the ‘correct’ class, but is a consistent coarser classification, e.g., a *Port Scan Only* being classified as *Suspicious*. A source is inconsistently classified if its classification is both not the ‘correct’ classification and not in a consistent coarser class, e.g., a *Port Scan Only* is classified as *Normal*.

Figure 4 illustrates the performance expected of the rule-based CISA IDS in terms of these four comparison categories defined above as data is streamed into the system at increasing rates. Rate is measured in terms of connections per second from all firewalls. Correct performance of the CISA is characterized by a gradual shift of sources from the first category to the third category as the rate increases with only a small percentage of the sources being categorized as the fourth category (inconsistently classified) at any rate.

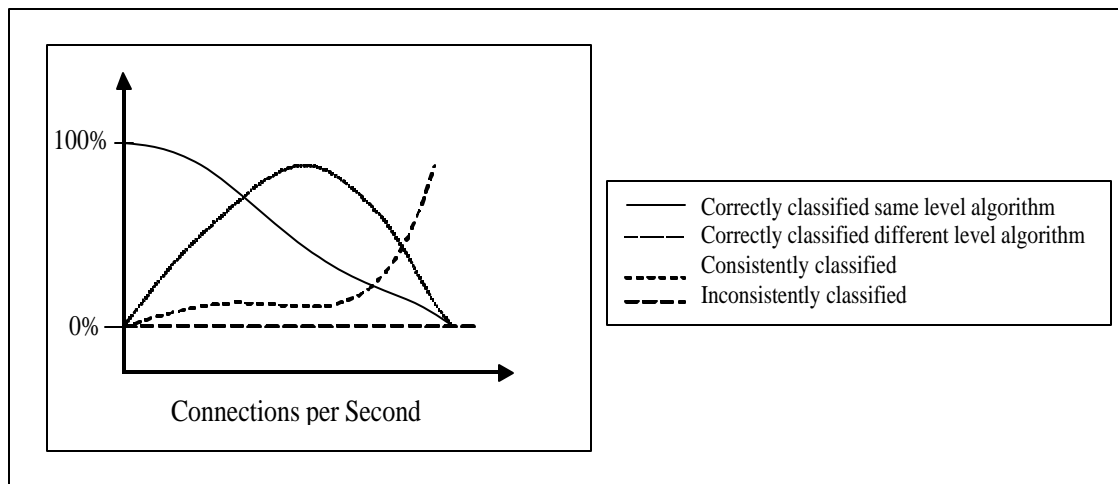


Figure 4. Expected CISA Performance as a Function of Stream Data Rate

The actual performance of the rule-based IDS CISA based on a prototype implementation is presented in Figure 5. In general the observed performance is consistent with the expected performance illustrated in Figure 4, with two exceptions. The first is that the “Correctly Classified with same level algorithm” percentage does not drop to zero as the rate increases. This is due to about 28% of the sources belonging to the Level 0: *Too Early to Tell* group (too few connections to compute any valid feature) which will always be classified correctly. The second exception is that the “Correctly classified with different level algorithm” peak is less than anticipated. This is an artifact of the way in which the simulation was conducted. In particular when performing this test, we were using proof-of-concept code segments to

verify the CISA algorithm concept which were not optimized to handle variable flow rates. The actual experiment streamed the entire dataset in at once and gave the system varying amounts of processing time. While conceptually similar, this approach alters the prioritization scheme putting too many high priority 'data storage' requests in the data storage object queue, so that lower priority feature calculation requests do not get to the top as frequently.

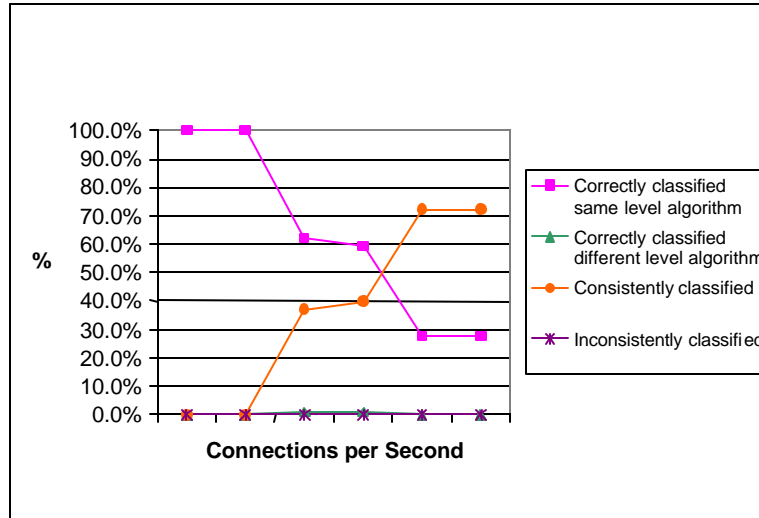


Figure 5. Observed Rule-Based CISA Performance

Dynamic CISA

The second CISA algorithm is dynamic and does real-time classification of source data objects based on cluster analysis. The clustering criterion minimizes the sum of squared distances from cluster means (also called centroids). Clustering is done with a variant of K-means clustering called **nearest centroid sorting** (see [1]). Nearest centroid sorting methods choose seeds by taking a preliminary pass through the data. The process starts with K seeds (typically the first K points) which are trivially the centroids of K clusters of one point each. The algorithm takes each subsequent data point in turn, adds it to the cluster associated with the nearest centroid, and then recalculates the centroid. During this process if centroids get too close together, they are merged. If a data point is too far from all current centroids it becomes a new centroid (of a cluster initially containing one point). The centroids that result from this initial scan are used to seed the K-means algorithm. This allows the data to determine the number of clusters, but retains the speed of K-means over hierarchical methods.

Little change is expected in clusters with the addition of a new point or the update of the feature set for one point. This is exploited by using the centroids from one analysis cycle to seed the K-means process the next time cluster analysis is performed instead of using default seeds. This approach was used in the prototype due to resource constraints dictating use of off-the-shelf tools. More sophisticated approaches such as those in [6], [8], and [11] would be natural refinements.

A current cluster assignment and priority are provided to each source data object every time the algorithm updates. Priority is based on the distance of an observation from its associated centroid, with observations further from their centroids given higher priority. Thus the more uncertain a classification is (i.e., farther from its centroid and the closer to other centroids), the higher its priority. The current behavior of each cluster is continually characterized by the behavior of the cluster's representative centroid.

This algorithm was tested by processing a stream of several hours' worth of firewall log data and monitoring the cluster analysis results. One of the strengths of an unsupervised method such as cluster analysis is that it characterizes behavior into groups based on what is currently present in the data, not based on what has been seen previously however perceptively by experts. This gives it the flexibility to identify new behaviors as they emerge, but puts the onus of interpreting cluster meaning on the user.

Graphical presentations of cluster results can be useful in interpretation. Figures 6 through 8 illustrate three of these which represent snapshots of the cluster analysis results taken at two points in time.

Figure 6 shows the clusters in “First Two Principle Component” space, similar to what was shown in Figure 3. This type of display can be intriguing to watch as clusters move, merge, and calve, but has limited use as it provides no interpretation of the clusters. One important use, however, is the rapid identification of extreme behavior. In the plot at 45 minutes (Figure 6), the black dot is an outlier cluster consisting of one point (Cluster 0 is a cluster of outliers). As we continue to watch the display, a point in Cluster 4 distinguishes itself by moving away from its cohort. In the plot at 85 minutes (Figure 6), its separation from its peers is clearly visible.

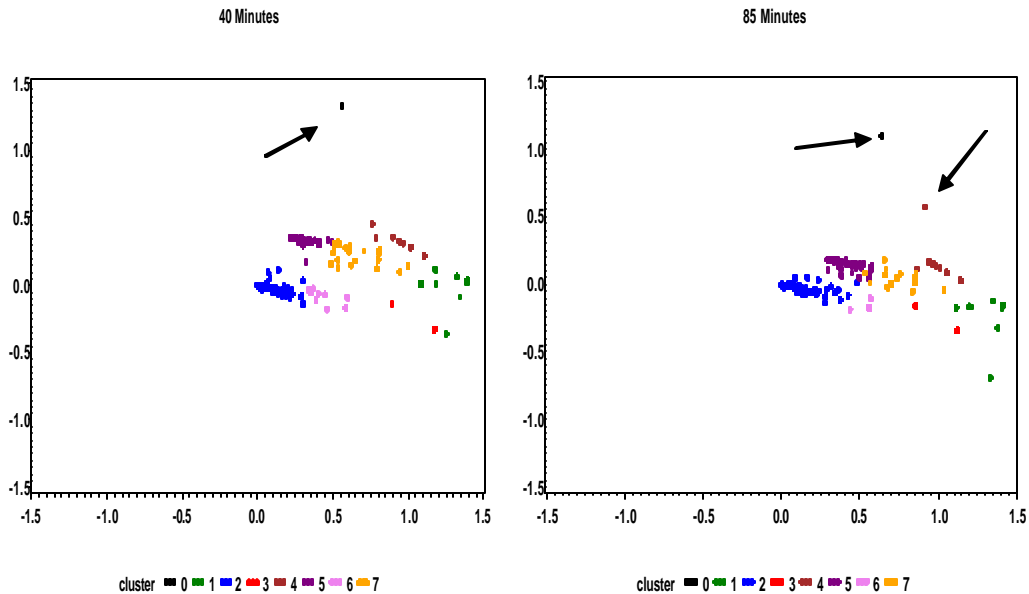


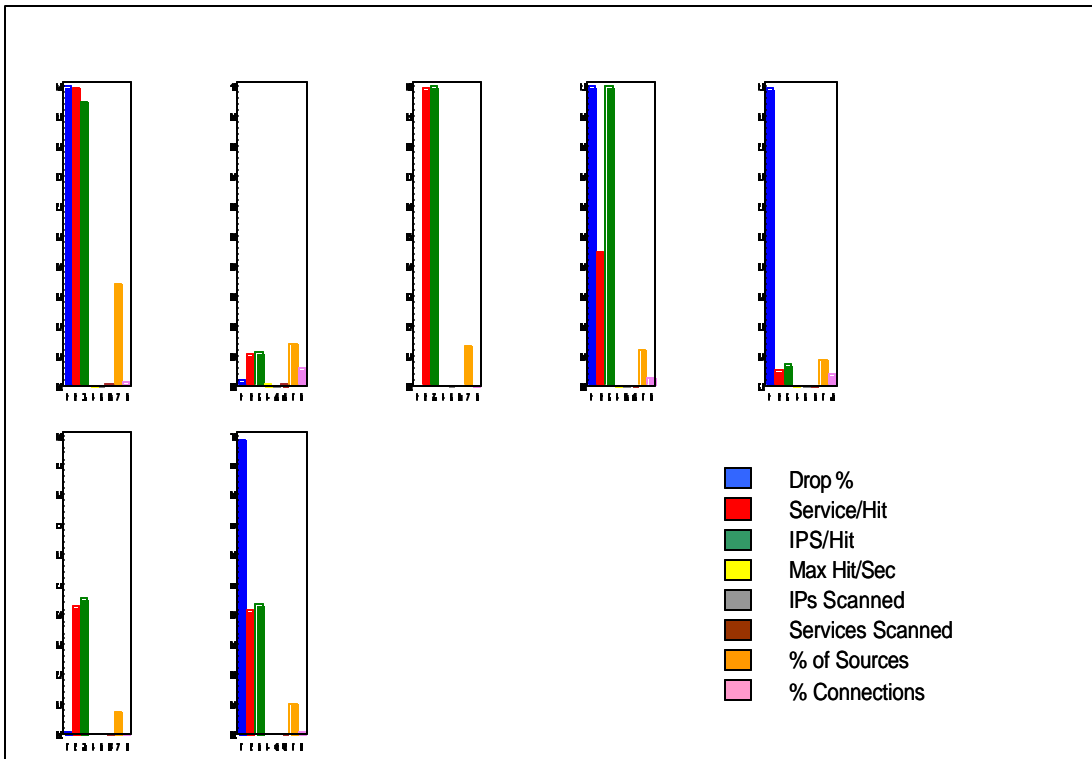
Figure 6. Clusters at Two Points in Time

Figure 7 presents a “dashboard” display of the cluster means or centroids. Each cluster in Figure 6, with exception of the outlier cluster (Cluster 0) has a vertical bar chart. Each bar represents the mean value of a feature for the cluster and all features have been standardized to a [0,1] scale. Added to the standard feature set are the percentage of source IP addresses making up the cluster and the percentage of overall connection attempts generated by sources in the cluster.

The blue bar represents the percentage of connect attempts that were blocked by the firewall. Clusters 1, 4, 5, and 7 all have drop percentages near 1 and are clusters that probably represent some form of malicious usage. The red and green bars represent ratio measures, namely the number of distinct ports requested per hit and the number of distinct IP addresses requested per hit. If the number of distinct ports requested per hit is 1, then each connection attempt is requesting a different port. If, in turn, there are many connection attempts for that source, a port scan is indicated.

We observe very high ratio measures in Clusters 1, 3, and 4, and fairly high values in Clusters 6 and 7. We further note that Cluster 1, 4, and 7 also have high drop percentages, but Clusters 3 and 6 do not. Looking carefully at the other measures we note that the percentage of sources represented in Cluster 3 and 6 is moderate (Orange bar), but the percentage of connections is extremely small (Pink bar). We infer that Cluster 3 most likely represents sources that made one connection attempt and accessed one port, hence had a ratio of one. Similarly Cluster 6 most likely involves several connection attempts on the same port. These behavior profiles are likely not of concern. Clusters 1 and 4 have observable connection percentages and probably represent sources which are conducting port and/or IP address scans.

45 Minutes



85 Minutes

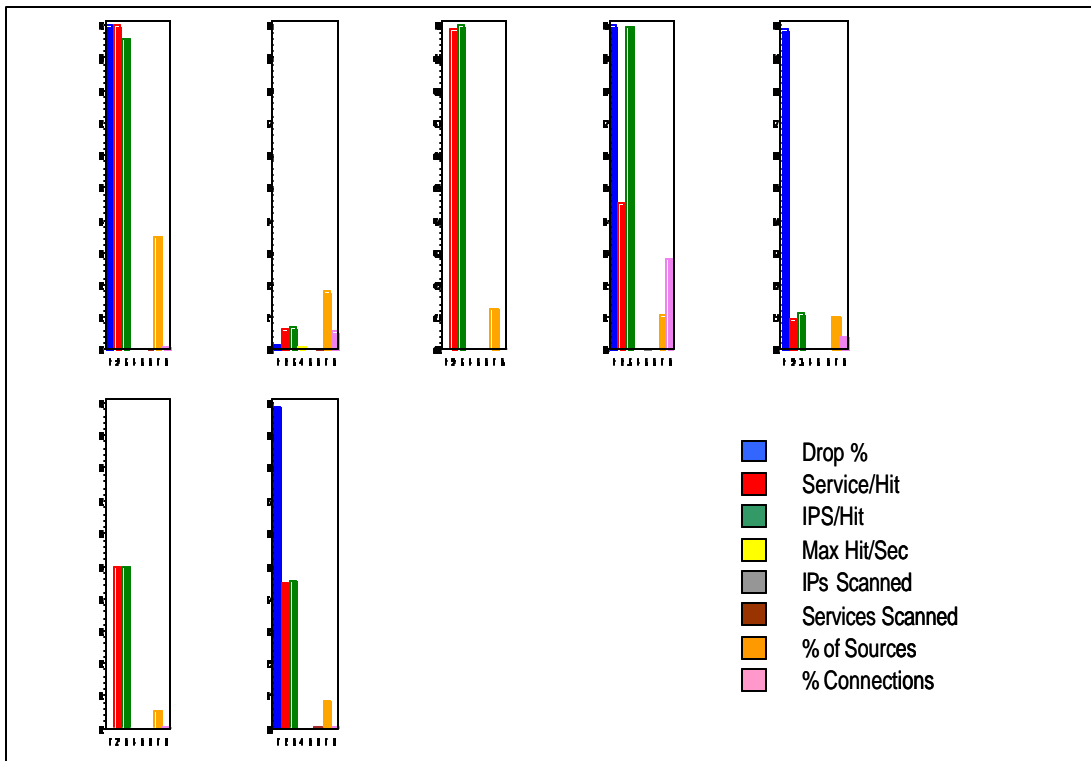


Figure 7. Cluster Profiles at Two Points in Time

Cluster 7 also does not have a visible connection attempt rate on the chart and likely represents individuals who attempt to connect a time or two and were dropped by the firewall. They either gave up or changed their IP addresses, which makes this a group worth monitoring. Sources in Cluster 5 are being dropped from the firewall most of the time, are making a fair number of connection attempts, and are not doing port/IP scans though we cannot identify what they are doing from the available data. Richer data with a richer feature set might answer this question. Finally Cluster 2 consists of sources that are rarely dropped, access a small number of ports or IP address and have a fair number of connections. This appears to be a dominant mode of “Normal” behavior.

We note that the no clusters have a high value for the number of services scanned, the number of IP addresses scanned, and the number of hits per second. Also the percentage of connection attempts does not sum to 100 across the clusters. The reason for the first observation is seemingly that while we see extremes on these measures at an individual level, we are not currently seeing this in the aggregated cluster behavior. Recall also that these clusters do not include the outlier (Cluster 0) which accounts for a high percentage of connection attempts, which answers the second observation. We address outliers next.

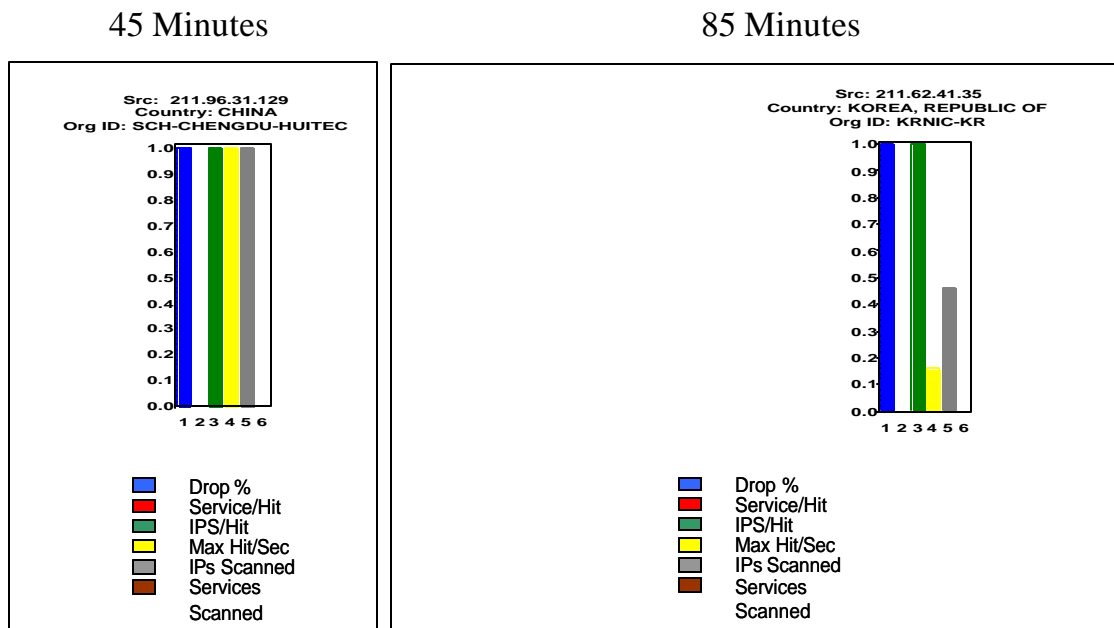


Figure 8. Outlier Drilldown at Two Points in Time

The CISA system stores data on individual sources so we can drill down at any time to examine the features of an individual source. Figure 8 shows the profile bar chart for the single outlier in Cluster 0 at the first point in time and for both this source and the source that is distinguishing itself from Cluster 4 at the second time point. A *whois* lookup is performed on these two sources and we find that one is from China, the other from the Republic of Korea. The source from China, which was seen as an outlier early, is scanning IP addresses only (has both a high IP/Hit ratio and a high absolute number of addresses scanned) and is attempting to access at a very high rate (from other analysis we determined it to be about 3000 connection attempts per second). This single source is responsible for so many connection attempts that the percentages shown in Figure 7 are small, tallying to just over 10% at the first time point. The Korean source is connecting at a fast rate relative to most sources, though significantly slower than the Chinese source. It is high enough, however, for the percentage of connections associated with Cluster 4 at the second time point to approach nearly 30%. If we observe the profile bar chart for the Korean source we

observe the bar representing the absolute number of IP addresses scanned (Gray bar) to steadily rise. Concurrent with this, the distinguished point in Cluster 4 in Figure 6 is seen to move towards the black dot representing the Chinese outlier, indicating that both sources are displaying similar behavior profiles.

Conclusions and Future Directions

This work has defined a new architecture and type of algorithm for managing stream data in a manner which smoothly handles the trade-off between decision making/characterization algorithms that store and process data quickly enough to keep up with the flow of stream data and algorithms that provide a sufficiently precise result. The approach scales automatically to optimize accuracy/precision as a function of data flow rate, improving the precision/accuracy when there is more processing time available relative to the rate of data flow.

To date CISA has been validated in pieces with prototype and proof-of-concept code. Through this process, the value of the CISA was made evident. At the same time a number of technical performance issues involved with object communication and data storage arose. The logical next step is to resolve the technical issues and move beyond the proof-of-concept pieces to more mature and unified demonstration code. Though this first instantiation of CISA was carried out within the context of intrusion detection, the ideas apply broadly and applications in other domains will be sought to implement these concepts with a different set of algorithms. Finally the applications of an evolving cluster analysis in the context of a stream environment are compelling. The work to date, being proof-of-concept, used existing commercial algorithms in standard packages. A third initiative is to develop the stream clustering capabilities of CISA using state-of-the-art stream clustering algorithms.

References

- [1] M.R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York 1973.
- [2] S. Babu and J. Widom. Continuous Queries over Data Streams. *SIGMOD Record'01*, September 2001, pp. 109-120.
- [3] B. Babcock, S. Babu, M Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. *Proceedings of the Twenty-First ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Madison, Wisconsin, June 2002, pp. 1-16.
- [4] P. Bonnet, J. Gehrke, P. Seshadri. Towards Sensor Database Systems. *Proceedings of the Second International Conference on Mobile Data Management*, Hong Kong, January 2001, pp. 3-16.
- [5] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring Streams – A New Class of Data Management Applications. *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002, pp. 215-226.
- [6] M Charikar, C. Chekuri, T. Feder and R. Motwani. Incremental Clustering and Dynamic Information Retrieval. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 626-635.
- [7] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and Mining Data Streams: You Only Get One Look, *Tutorial at 2002 ACM-SIGMOD Int. Conf. on Management of Data (SIGMOD'02)*, Madison, WI, June 2002.
- [8] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. *Proc. IEEE Symposium on Foundations of Computer Science (FOCS'00)*, Redondo Beach, CA, pp. 359-366, 2000.
- [9] W. Lee, Applying Data Mining to Intrusion Detection: The Quest for Automation, Efficiency, and Credibility. *SIGKDD Explorations*, Volume 4, Issue 2, 2002, pp. 35-42.
- [10] S. Muthu Muthukrishnan. Data Streams: Algorithms and Applications. *ACM SIAM SODA*, January, 2003.
- [11] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Mining and Knowledge Discovery*, Volume 1, Issue 2, 1997, pp. 141-182.