Intelligent Avionics with Advanced Clustering

John Meier
Boeing Corporation
Saint Louis, MO, 63166, USA
Email:john.l.meier@boeing.com

Todd Sproull and G. Adam Covington and John W. Lockwood
Department of Computer Science and Engineering
Washington University in Saint Louis
St. Louis, MO 63130 USA
{todd,gac1,lockwood}@arl.wustl.edu

Abstract—In this paper, we consider tracking targets using multiple distributed sensor platforms. Rather than sending the tracks to a central location, such as a command and control center where information is exchanged between platforms, we consider a distributed solution. While fixed position single sensor tracking of a single target is considered straightforward, multiple sensors on the different platforms with overlapping coverage is complex because duplicate tracking data is generated for the same targets. Redundant information generates network messages that in turn overload the network performance, and may result in traffic congestion on limited avionic bandwidth wireless links that prevents time critical data from reaching its destination. In our approach, we identify similar tracking data to be distributed and send only one copy of the message.

In order to identify redundant data, we use a clustering algorithm to evaluate the large volumes of sensor information. Distributed multiple target tracking (MTT) combines track observations from different sensors to identify the same target. Massive computation and communication is required for distributed real time MTT. In this paper, the K-means clustering algorithm is used to aggregate redundant tracks. Software simulations using Matlab and emulation tests using Emulab show significant improvement of the information quality by using clustering. An MTT system was prototyped with FPGA hardware to cluster high volumes of data with low latency in real time at the network layer.

TABLE OF CONTENTS

This work was funded by a research grant from the Boeing Corporation.

IEEEAC paper# 1669, Version 4, Updated October 20, 2007

1. INTRODUCTION

Commercial and military aerospace use multiple sensors on a variety of platforms today to track targets. Sensor data is partitioned into sets of observations, or tracks, that provide a time and distance history of target location. Track data is then processed to identify the number of targets and identify key metrics including velocity, future predicted position, and target type. Avionic networks interconnect numerous sensors that generate a large volume of data. Communication bandwidth is a valuable resource that must be used wisely to track targets effectively. Multiple Target Tracking (MTT) must locate and identify targets quickly.

Existing MTT detection, classification, and tracking algorithms work well on a single platform. Centralized data fusion collects data from multiple sensors. Rather than implement another centralized algorithm, we consider techniques for distributed algorithms to regulate communication for distributed track fusion.

Multiple platform sensor MTT systems need to gate and correlate a target's parameters (position, range rate, velocity, and acceleration) [1]. Our approach uses clustering to associate MTT data for distributed tracking based on scenarios such as the one depicted in Figure 1. We evaluate the cost and benefit of distributing track data between nodes with sensors for improving situational assessment (SA). Track data (observations) are distributed using shared bandwidth between platforms to improve SA. Clustering improves SA by providing improved data association. Clustering results in prioritization of network data, conservation of bandwidth and lowering the track fusion latency. Traditional methods to distribute track observations often load the existing bandwidth beyond the channel capacity resulting in information latency and loss. Figure 1 illustrates that multiple sensors report their observations of every target within range (depicted as circles). Our scenario has multiple targets moving in defined patterns within range of multiple sensors to assess scalability.

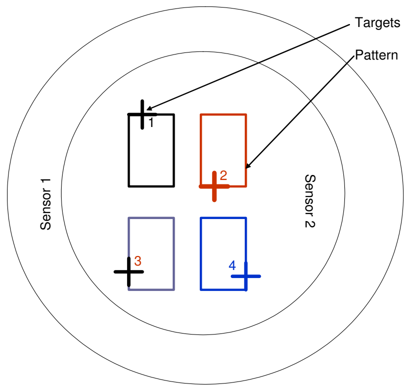

Figure 1. Targets are simultaneously tracked by multiple sensors

Real time SA requires improved dynamic exchange of sensor observations which often generates duplicate data for targets located in overlapping coverage which cause network overload. It is critical to select only the best information to send over the limited bandwidth. Lossless compression techniques are not effective in fitting all the information within the limited bandwidth for large scale systems. Clustering intelligently groups track messages autonomously in real-time to use available bandwidth with the highest priority track data. Our goal is to reduce the number of tracks exchanged while retaining information quality. The effectiveness of using our clustering algorithm operating in a distributed environment was proven using large-scale Emulab experiments.

2. Clustering and Classification

Clustering algorithms help to group data at discrete update points. The measurement of the data to be clustered is defined as the distance metric. We evaluated Manhattan, Euclidian, and Chebyshev distance algorithms for use with the K-means clustering algorithm. Our distributed MTT clustering algorithm uses Manhattan distance with a modified K-means algorithm to find centroids of the clusters of tracking data.

Classification of data identifies which data is similar when prior information about a centroid is known. A centroid is the representation of the target's predicted position based on the density of targets. The centroid is adjusted at discrete times to predict the target location. Track data is sorted and clustered for each target. Target data is stored in a table as shown in Figure 2 as the track state for clustering. Observations received at different times (t1,t2, t3,...) are compared to the projected centroid of each track or cluster stored. Velocity and position data is used to associate the new observations received with the appropriate track or clusters stored.

Our MTT track clustering algorithm calculates the distances between tracks and projected target positions associated with the tracks (centroid). The algorithm identifies whether the incoming track maps to an existing centroid then either updates the centroid or creates a new cluster for the incoming track. Finally, the algorithm then determines if the track should be used only at a local node or sent to other nodes.

The traditional K means algorithm requires specifying the number of clusters. Since we never know exactly the number of targets being tracked, applying traditional clustering algorithms is impossible. We evaluated multiple clustering and distance algorithms for improved track data association and bandwidth matching using Matlab. The thresholds were adjusted to match the MTT track gating. The results provided clustering design and development constraints used in simulation, emulation and hardware evaluation.

To minimize the latency required to cluster data and to maximize throughput of the target tracking algorithm we implemented a clustering algorithm in Field Programmable Gate Array (FPGA) technology to parallelize the processing. The main three hardware modules that were prototyped include: Track Cluster, Time Compare, and Update. The Track Cluster module calculates the Euclidian distance and maintains a list of current clusters/tracks. The Time Compare module determines the priority and whether to send or aggregate the data. The Update module computes the projection of the target (centroid) for clustering the track data and passes the projection of the Track Cluster module to be stored. The prototype modules are demonstrated using an open platform called the NetFPGA. The NetFPGA processes real time MTT data as it is received in real time over a network. Hardware achieved a significant improvement over traditional software processing methods. Our parallel hardware design can perform 4 simultaneous distance metric measurements with up to 100 simultaneous tracks operating at a clock speed of 125 MHz. The total time required for distance calculations, assignment determination, and updates requires 0.904 μs for each incoming track which creates a real time throughput of approximately 1.1 million packets per second.

3. Related Work

Target tracking includes data association and track filtering. Data association receives observations and must assign them to existing tracks. Correct association is difficult because the targets may be closely spaced together or located between tracks. Sensor measurements of the targets may be imprecise due to measurement resolution, noise, and other error sources. Many data association algorithms have been presented [2] [3] for group clustering and target clustering but few leverage recent advancements in data clustering tech-

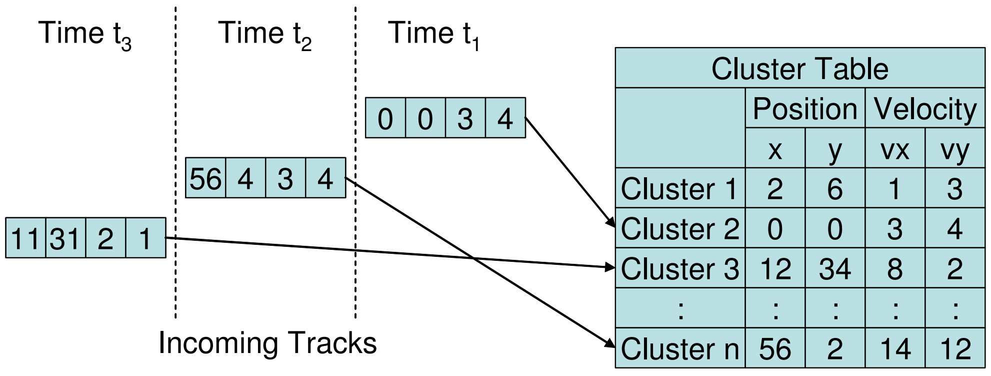

Figure 2. Tracks mapped into L dimensional vectors are clustered into groups of current tracks

nology for improving bandwidth usage. Older methods use Nearest Neighbor (NN) algorithms that make decisions as the data arrives while newer methods delay decisions by storing the data used in Multiple Hypothesis Tracking (MHT). Our clustering methods use NN algorithms for data association. Modern radar systems often use the Suboptimal Nearest Neighbor (SNN) algorithm while the Global Nearest Neighbor (GNN) has recently been proposed. Often multiple closely spaced aircraft are grouped together and mistaken for a single target. The GNN approach has demonstrated very reliable results for modern radar systems when contrasted with SNN.

Clustering has been used in two ways for tracking in the literature. Target clustering groups similar data elements or observations together to form a track (usually without prior knowledge). Group clustering typically computes a location and velocity centroid of a large number of closely spaced targets moving in the same direction in order to reduce the track data transferred, system loading, and miscorrelation. Target clustering is the application we are addressing.

Target clustering automates the target track gating and distribution of tracks. Current radar monopulse tracking methods have trouble handling multiple unresolved targets within the beamwidth which creates distortion by averaging the target measurements into a group centroid. Proper setup of thresholds improves correct classification and sorting of data to reduce miscorrelation. Clustering algorithms use similarity measurements known as the ”distance.” The distance is computed from observations to centroids. Normally, centroids represent the average of grouped data elements however we project the last observation to the temporal-space for the discrete time interval needed to match the bandwidth available.

Clustering applied to distributed fusion using avionic networks is a novel application. Image and text based clustering was the basis which formed our current approach. The two main forms of clustering are agglomerative (bottom-up) and divisive (top-down). Each approach utilizes a distance metric that measures the difference between two elements. Agglomerative clustering treats every data element as a separate cluster and merges clusters if they exhibit similar distance metrics. Divisive clustering starts with all data elements in the same cluster and partitions them into different clusters based on the distance metric. Both forms of clustering are described in [4] as it applies to hierarchical document clustering using the K-means algorithm.

The original K-means algorithm was described by Duba and Hart [5] and has been used by Estlick et al. [6] and Lesser et al. [7] to implement a hardware approach of K-means clustering for hypersectral images. They [7] compared three different distance metrics that included: Manhattan, Euclidean, and Max distances. They determined that the Manhattan distance would be the best fit for their hardware. Covington et al. [8] showed the implementation of K-means mapped into integer arithmetic. They utilized a Cosine Theta distance metric since this metric (also known as the spherical distance) provides a better distance in high dimensional sparse data.

Spectral Clustering is another algorithm that has been used to cluster data vectors. This algorithm provides an effective method to cluster sparse high dimensional data sets. A spectral clustering algorithm described by Ng et. al. [9] calculates the top features or dimensions to cluster by calculating and selecting the top eigenvectors to cluster. The heart of the algorithm described used K-means to cluster the top eigenvectors.

4. TRACK CLUSTERING ALGORITHM

Most clustering algorithms operate on a fixed set of data and are commonly iterative. Every cycle a data element is selected and is moved to the cluster that is determined to be the best fit by the distance metric. The clustering algorithm used to cluster track data differs from these cyclical algorithms. Since the clusters represent known tracks that could move, the incoming track data can only be clustered once. The track clustering also utilizes a cluster threshold to determine if a track is close enough to be included in a cluster. This is an important addition to the algorithm. The cluster threshold allows the creation of new clusters when tracks are significantly distant from existing clusters.

The algorithm for track clustering operates as follows:

1. Calculate distances from incoming observation to centroids.
2. Determine if there is a centroid C_{min} close enough to have a distance below the cluster threshold.
(a) If C_{min} exists, assign track $\vec{t}$ to centroid and update the position of C_{min} based on the velocity.
(b) If C_{min} does not exist, add the track $\vec{t}$ as a new track.

Although there are numerous distance metrics that can be used for clustering, this work utilizes the Manhattan distance metric.

Manhattan Distance

The Manhattan distance, or city block distance, provides the absolute distance between two data elements. In a two dimensional grid the calculation totals the absolute value of the difference between the vector elements. Given the example shown in Figure 3, the Manhattan distance is nine.

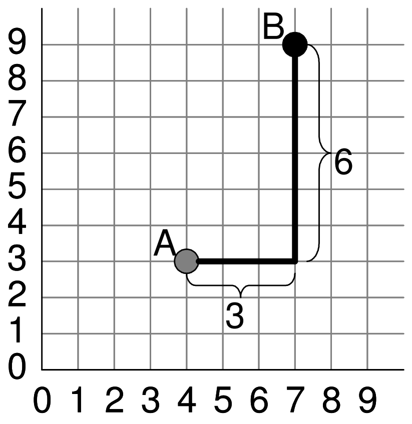

Figure 3. Manhattan Distance calculation in two dimensions:
$|A_x$ - $B_x|$ + $|A_y$ - $B_y|$ = |4 - 7| + |3 - 9| = 3 + 6 = 9

$$ManhattanDistance = \sum_{i \in 0..N-1} |\vec{A}_i - \vec{B}_i| \qquad (1)$$

5. ANALYSIS

The major challenge faced by distributed multi-target track (MTT) fusion is choosing the right information to send at the right time over severely limited bandwidth links to construct a scalable unified picture that will enhance situational awareness (SA). MTT employs one or more sensors, together with computing resources, to interpret the environment based on a series of measurements. MTT partitions sets of measurements (observations) or tracks for object representations in space and time. Target prioritization (missiles, aircraft, trucks, ships) is critical to correctly assess the environment in real time. This research creates a more intelligent interface between the application (fusion) and the network (distribution of observations) to reduce latency while increasing the value of information transferred. Fundamentally we reduce latency and preserve bandwidth by adding intelligence at the network layer that is able to make real time decisions.

The future SA technology trends use distributed fusion to increase track accuracy and reduce latency in acquiring a common operating picture. Today centralized fusion is usually accomplished at the application level and relies on the network to transfer all observations. The fusion application has very limited visibility into the network layer. Operating at the network layer more effectively is critical to distributing the many observations required for centralized fusion or distributed fusion. Distributed fusion runs multiple copies of the fusion algorithms at local nodes (near the edge) of the network and relies on efficient transfer of the right data.

The problem is difficult due to the many constraints such as unreliable wireless transport, limited processing power at the edge, and the use of multiple legacy wireless communication links with low throughput. There are many solutions devised to overcome unreliable transport in wireless networks however the associated overhead and latency limit their effectiveness. Limited edge processing makes running the fusion algorithms at distributed nodes difficult. Typical avionic wireless links offer less than 100 Kbps while the offered load for large distributed fusion is 100 Mbps or more. Large packet latency or out of sequence packets often result in information deletion by the fusion algorithms at the application layer. Recognition of the inappropriate transfer of data will save bandwidth and result in less processing cycles wasted. We hypothesize that better use of network resources can help to improve distributed fusion performance.

Theoretically, each distributed node uses an identical fusion engine to evaluate every track observation which should generate exactly the same SA picture. We realize that it is impossible for every node to be sent and process every track observation to develop exactly the same operation picture. Reduction of data and preservation of the required processing cycles is critical to developing a realistic solution for improving SA.

Two main types of distributed track fusion messages are used to initialize and update current tracks. There is a time during initialization where multiple distributed nodes have separate identifiers for new tracks detected however the goal is to quickly converge to a unique identifier used by all sensors on different platforms tracking the same target. Each track should be assigned a unique identification (ID) soon after initialization occurs. Our solution does not depend on a unique identifier assigned but will use it if provided. Distributed track observations also contain position, velocity and relative time of the measurement information.

There are scalability problems with distributing every track observation to all nodes. Recently the MTT application tries to perform network layer functions at the application layer. This approach creates real time performance issues, demands significant increases in processing, presents optimal bandwidth usage challenges, limits key access to network management parameters, and adds latency due to large queues buffering network packets. Our novel research evaluates using clustering to reduce or eliminate these problems.

Traditional information theory was evaluated to help map the track observations to queuing decisions for more efficient bandwidth allocation. In our approach, the goal is to identify similar tracking data to be distributed and send only one copy of the message. Shannon [10] discussed "GEOMETRICAL REPRESENTATION OF MESSAGES" to represent messages in multiple dimensions more efficiently. Eliminating message redundancy can happen by simply recognizing different information in the same message that is not necessary for recreation of the information being transferred. Shannon proposes improved mapping of higher levels to lower levels of communication for more efficient transfers. Intelligent reconstruction by the source and receiver can provide the improved mapping by eliminating messages or information normally sent. In the case of distributed tracks, the fusion algorithms are insensitive to a certain amount of variation in distance, known as tracking error [1]. Mean and variance of the tracking error are used for calculating the dispersion matrix (variance and covariances of x,y positions of members) for a group of track observations. MTT uses variations of the dispersion matrix to identify the associated tracks (estimate the position of target). We evaluate the impact of improvements to the lower level queuing using higher level geometric methods, similar to the example provided by Shannon [10]. We evaluated changes relative to the true track path or shape generated relative to the track constructed from the data sent over bandwidth limited links. The challenge is to provide the information required for accurate tracking while selectively eliminating redundant or unnecessary information to match the bandwidth capacity available. Our solution adds intelligence between the source and receiver to make difficult real time decisions at the network level to increase the value of information and improve network quality of service (QoS).

Messages differing by only a slight variation of measurement (to a limited extent) represent nearly the same information and offer an opportunity to eliminate redundant information. This may also reduce the number of dimensions in the current message space. Track observations which are considered equivalent by the destinations (receivers of track observations) can be grouped together and treated as one point or a reduced set of key observations. Equivalency is evaluated by assessing the relative closeness of the observation to the projected centroid. We propose that this grouping method requires fewer messages to specify one of these equivalence classes defined as we cluster the track observations rather than sending sequential non-prioritized observations. For example, we use a two-dimensional space to indicate the value of this method by comparing normal (tail drop) network queuing with clustering. Simulated targets are controlled and fly in a square pattern of known dimension to simplify assessing any perturbation observed. The deviation in target tracking shape (represented by the series of observations) are quantified relative to area differences of the shape. We compare the known area generated by the simulation (target flying in a square) to the area constructed based on the limited observations sent.

Our clustering approach for distributed fusion uses both spacial and temporal methods to evaluate equivalence. We adjust clustering thresholds to prioritize the information sent to other nodes. Distance methods, such as Manhattan or Euclidean distance, compare the distance from the centroid to the observations. If all observations are evenly spaced on a circle around the centroid, they can be regarded as equivalent, and theoretically can be reduced to a one-dimensional space or point. We extend this specific example representing points (track observations) within the circle as key messages to be sent at higher priority. The radius of this circle is defined by the clustering threshold. We preserve state by storing each target's most recent track observation sent for centroid projection.

The spacial method described above clusters based on distance. We first project the centroid using information shown in Figure 2 and the following equation:

$$Centroid = stored(X,Y) + (timeinc) * (currentvelocity) \tag{2}$$

Second, the distance from the observation to the projected centroid is calculated. Third, if the distance is less than the clustering threshold, the observation is sent. Currently the clustering threshold is set based on empirical data however dynamic methods are planned for future implementations.

Temporal methods simply evaluate velocity to prioritize the observations and then select the correct update rates. The three priority levels are high (e.g. missiles traveling at Mach 2 or higher), medium (e.g. aircraft traveling between Mach and Mach 2) and low (e.g. trucks, ships traveling below Mach 1). We plan to use an adaptive weighting factor that will be adjusted relative to the proximity, target turn radius, and threat level represented by the target. Based on experience, the update rates for the high priority is set at 6 hertz, medium priority is set to 4 hertz and low priority is set to 2 hertz.

Our temporal algorithm compares the relative system time associated with last observations sent to decide whether the current observation should be sent. The update rate is varied based on the priority level of the target. The observation won't be sent unless the update rate set for the target has been exceeded, even if the observation is below the clustering threshold (close to the centroid). Currently the spacial and temporal algorithms operate independently but are sequentially dependent, as indicated.

We evaluated a severely constrained bursty link with a small tail drop queue (traditional network QoS) contrasted with a zero length queue (clustering) to assess effectiveness. Queue length helps to absorb variations in link bit rate but adds latency. Clustering adjusts the sending bit rate by using FPGA hardware to make real time decisions resulting in only very small hardware latency. Figure 4 below shows projections based on Matlab simulations of the improved performance offered by intelligent clustering methods contrasted with traditional queuing methods using graphical evaluation techniques described above. This illustrates that effective operation may occur with clustering at much lower bandwidths than traditional methods. Figure 5 illustrates a modified target tracking pattern difference due to loss of critical data. The area represented by the reduced data set has reduced area as expected. Significant area reductions occur depending on the randomness of the queue, burstyness of the data, and redundancy of the target pattern.

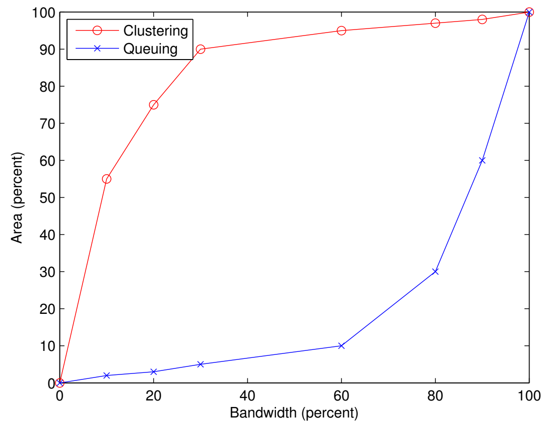

Figure 4. Value of information example

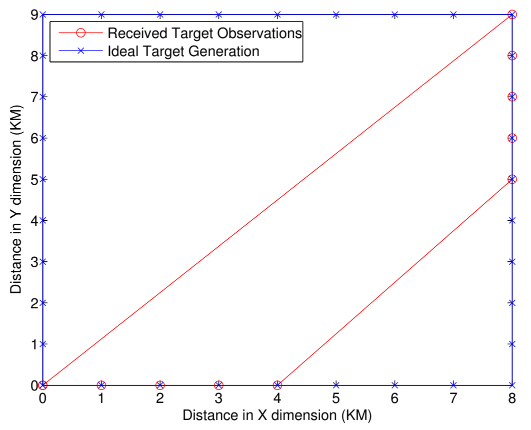

Figure 5. Area difference example representing information value

The queuing solutions today are widely used. Our Matlab simulations take into account the bursty nature of wireless links and the loss of critical updates for short periods of time as noted in Figure 5. The ideal target pattern shown as a square (true representation) is represented as a distorted parallelogram to illustrate the change in area. The representation remains distorted until a large percentage of the bandwidth required is available as illustrated in Figure 4. The clustering solution is shown to provide increased information content with reduced bandwidth and latency.

6. Experimentation

Software Clustering

This section describes experiments to determine the latency related to available link capacity and use of software to cluster the track observations. The first set of experiments investigated the amount of latency experienced by a single node receiving track data. This experiment inserted a time stamp on packets leaving the sender. The receiver then compares the packet time stamp against its current time. The network time protocol (NTP) was used to synchronize clocks on all machines in the experiment.

Each node sent sensor track data to all other nodes in the network using IP Multicast. A track generator application provided by Boeing created the tracks with the flexibility to generate diverse patterns or shapes. The software creates targets of interest and records their position as they change over time based on the pattern selected. Modifications were made to the track data generation software to provide simple X, Y coordinates and X,Y velocities. The track data also contained an identifier field describing the type of track data and packet length.

Figure 7 contains the graph demonstrating an increased latency experienced at the gateway (GW) node for track data. The gateway node has the ability to process the data at the network level between when it is received at the input port and routed to the output port. As the networks grows from 10 to 68 nodes, the amount of track data increases as well as the latency for a single track to be processed. The maximum latency varied from 109 to 260ms as the number of nodes increased. The average latency ranged only from 103 to 109ms.

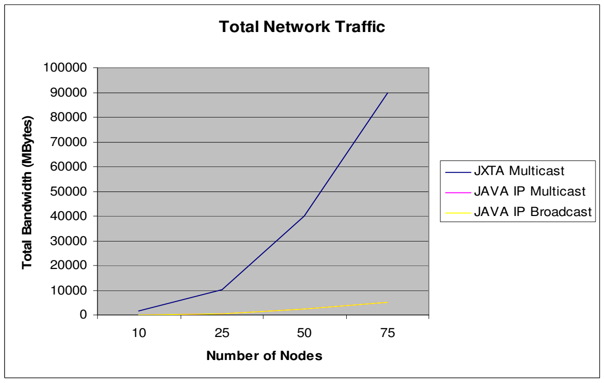

Figure 6. Bandwidth Costs for Distributing Track Data with Different Protocols

In the next experiment, clustering algorithms are proposed for both software and hardware.The nodes topologies consisted of 25, 50 and 75 nodes, each sending track data to all nodes in the network, including a GW clustering node. The GW

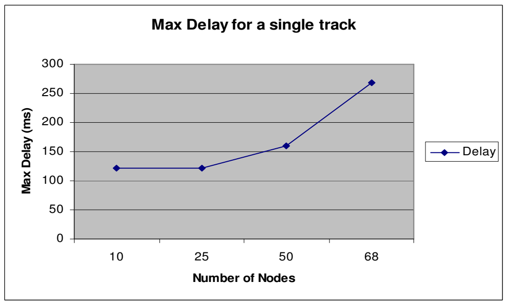

Figure 7. Maximum Latency to Receive a packet over a 100ms link

clustering node uses a K-means clustering algorithm on each packet in order to identify similar track data. Each packet that arrives at the GW node is clustered and compared to packets previously clustered. If a packet is similar to a one previously clustered and recently forwarded to the neighboring network, it is considered redundant and discarded.

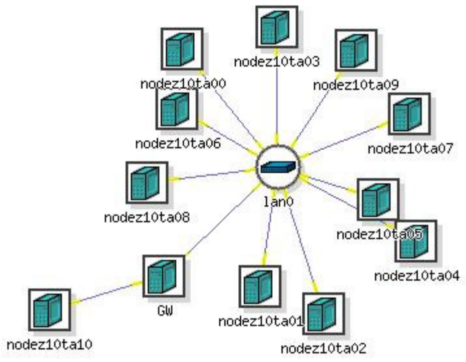

Figure 8. - 10 node Emulab Experiment with a Gateway Cluster Node and Neighbor Link

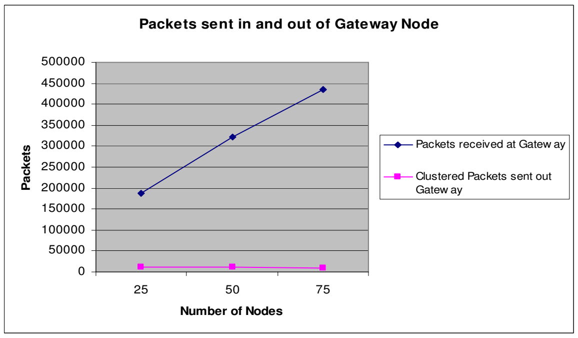

Figure 9. Performance of Clustering Algorithm in Software (Packets)

In Figure 8, the GW node connects to a neighboring network or node. Our experiments next deployed a single node connected to the other end of the GW with a 1Mbit/sec link had a latency of 100ms. Figure 9 illustrates the number of packets that the gateway node receives and clusters packets that are forwarded. The amount of packets it forwards varies from

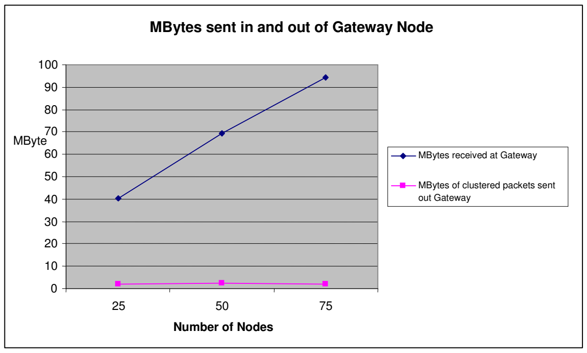

Figure 10. Performance of Clustering Algorithm in Software (Mbytes)

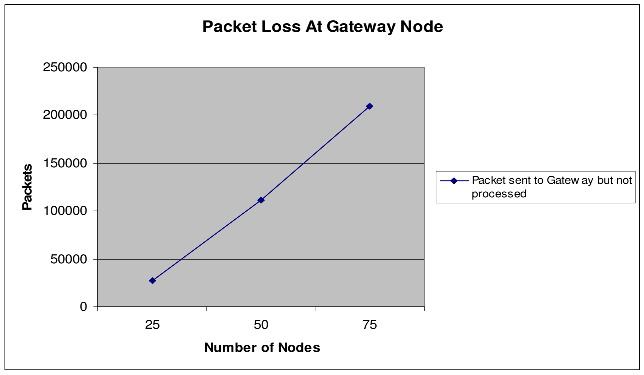

Figure 11. Packet Loss experienced at Gateway Cluster Node due to Software Clustering

1130 to 9400 packets. This is a considerable reduction in the network load because the amount of packets it receives and clusters ranges from 187,000 to 435,000 packets. Figure 10 displays the amount of bandwidth used in terms of megabytes processed in and out of the gateway node. Again, significant reductions in bandwidth are demonstrated using clustering algorithms. The incoming amount of data varies from 40 to 90Mbytes with the clustering gateway node reducing the traffic to 1.9 to 2.1 Mbytes. This provides an 18 to 48x reduction in bandwidth before traversing to the low capacity neighboring link.

The last experiment performed investigates the amount of packet loss experienced by the software clustering implementation. The clustering program was developed in JAVA and executed on a 3 GHz Pentium 4 CPU using a Linux operating system. The experiments utilized 100Mbit links with no additional link delay to provide the best case performance.

Packet loss is due to the slow sequential processing required for the clustering algorithm. Processing compares the clustered hash value of each incoming packet against the know hash values of the packets in memory. As bursts of packets arrived, the CPU was unable to investigate all of the packets before buffers overflowed. Since the track data is assuming the use of an unreliable communication protocol, no retransmission is issued for track data and it is lost. The hardware implementation of this clustering algorithm will not suffer the same speed limitations of software. Hardware maps the clustering algorithm onto an FPGA, that allows for significant parallelism. The results of the packet loss at the gateway node are presented in Figure 11. The number of packets that are unable to be clustered varies from 27,000 to 209,000 packets.

Figure 9 illustrates the amount of traffic in and out of the gateway node. The average packet size received at the gateway is 217 bytes. Each node generates 9,090 packets. In the 75 node example, 9090 x 75 = 681,750 packets are destined for the GW node. The GW was only able to receive 434,995 packets, and the others were dropped due to an overloaded CPU. The total traffic received at the GW was 217 * 434,995 = 94,393,915 bytes or approximately 94 Mbytes.

Figure 6 depicts the amount of traffic generated in the network as the number of nodes increased from 10 to 75. From this figure, the JXTA P2P Multicast protocol is the most expensive implementation to distribute traffic to all nodes. Using either IP Multicast or Ethernet Broadcast is significantly less expensive. From these results, the IP Multicast protocol was selected as the protocol to deploy in the software and hardware clustering applications. These simulations helped to demonstrate the problems of deploying the JXTA P2P at the Gateway Cluster Node. This protocol would create additional CPU and network load to further degrade the performance of the clustering. From Figure 6, the overhead associated with sending the same information using JXTA Multicast required 17 times the total bandwidth. This overhead comes from the additional complexity of pushing multicast (state information encapsulated with each message) using overlays to the application layer. Thus it is possible to predict 138,087,788 * 17 = 2,347,492,396 bytes or approximately 2.3 Gigabytes of traffic destined for the gateway node if P2P multicast is chosen as the distribution method. The GW node would be able to process significantly less if conventional hardware is used. Testing confirms that even a 3 Ghz CPU could not keep up with receiving and clustering of the smaller multicast packets which necessitates the use of FPGA or ASIC technology.

7. Hardware Architecture

Advanced SA requires the use of high speed networks with increased bandwidth to implement centralized track fusion to improve tracking performance. Distributed track fusion can process the data locally and only exchange key observations to improve SA. It is critical to select the correct distance and clustering methods to ensure the right data arrives at the right time to the distributed nodes.

Manhattan distance is used to calculate the distance to the projected target position or centroid. This ”coarse gating” technique is responsible for the majority of the data reduction. Track prediction is critical to setting the clustering threshold. Selecting the tracks to be clustered is a form of track-to-track gating. The time increment threshold selects the critical

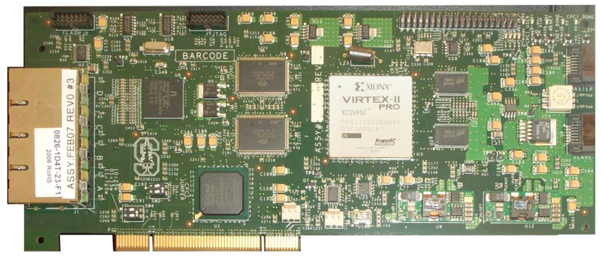

Figure 12. The NetFPGA platform used to implement Track Clustering

points at which the distance clustering is accomplished.

As each track observation arrives, the algorithms first extrapolates each stored source track forward using the velocity and time increment. Second, the distance to each projected centroid is compared to correctly associate the data with the correct track or cluster. If the distance calculated is less than the clustering threshold, the new observation will be selected as critical data for fusion. The update rate is also checked to ensure the most critical data is prioritized then sent.

Highly parallel state machine hardware implemented the algorithms above using the NetFPGA platform. The NetFPGA is an open source project that allows researchers to develop network applications and systems [11]. Developed projects include Secure Switches (Ethane) [12], Routers, and Rate Control Protocol (RCP) [13].

For the development of network solutions, the NetFPGA contains a Xilinx Virtex-II Pro FPGA. The board also contains a Double Data Rate (DDR2) SDRAM device, two SRAM devices, two serial ATA (SATA) connectors, and a quad-port physical-layer transceiver. The NetFPGA library provides a Verilog template for design that interfaces to the memory devices and the network interfaces [11].

8. Hardware Design

The track clustering algorithm performs four primary operations (1) calculating the distances between observations and centroids, (2) identifying if the incoming observation maps to an existing centroid, (3) updating the centroid or creating a new cluster, and (4) determining if the track should be dropped or sent based on the timetable. The hardware implementation is comprised of three primary modules: Track Cluster, Time Compare, and Update. In addition to these modules, control modules were needed to load and run the hardware clustering system. Figure 13 shows the architecture of the track clustering system.

Time Stamp

As the track data enters the system it is provided with a time stamp. The time compare module uses the 32-bit time stamp to determine if the track should dropped or sent based on the

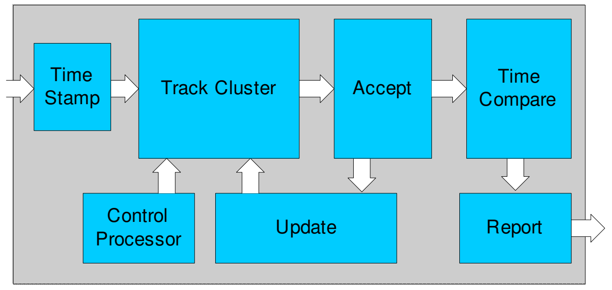

Figure 13. Hardware Track Clustering Block Diagram

defined update rates set by the time threshold.

Track Cluster

A set of smaller modules comprise the track cluster function. The first module calculates the distance and is designed in a modular fashion so that it can be replaced with diverse metrics implemented in hardware. The Manhattan distance calculation is replicated in parallel to evaluate multiple distances simultaneously.

The second component is the cluster table. The cluster table maintains a list of current clusters/tracks locally on the FPGA chip. The centroids of the clusters represent the projected position of the track for prioritization. This projected position determines if an incoming track matches a cluster. The calculated parallel distances are passed to the accept module to make the critical decision to forward the data.

Accept

The accept module compares all parallel distances to a cluster threshold. If the distance is less than the cluster threshold, the module will assign the incoming track to the specified cluster. The module will not assign the incoming track if the distances are greater than the cluster threshold. If this occurs, the system currently decides the incoming track is a new cluster and will specify that a new entry needs to be added to the cluster table.

Update

After the accept module determines if an assignment is accepted, the update module then updates the cluster table. The update is computed from the velocity information contained in the track data and determines the projected position of the track. The projected position is passed to the cluster table (in the track cluster module) for storage. The accept module determines whether the track is a new track and needs to be added to the set of known clusters. The update module will calculate the projection of the track and then send the information to the cluster table to be stored.

Time Compare

The time compare module determines if the track should be dropped or forwarded to downstream modules/systems. The module is comprised of a timetable that maintains two time values: the last time the cluster/track was reported and the last time a track message was received. A time threshold determines if a track message should be sent or aggregated. If the difference between the last sent time and the current time is greater than the threshold, the data is sent from the node. However, if the difference is less than the time threshold the data updates the cluster table however the data is not sent from the node.

The timetable also contains information regarding the velocity and the accuracy of the incoming track data. This data is used in conjunction with the time data to determine whether the system sends or drops the data. The velocity information is used to determine the three time categories or priorities of traffic: high, medium and low.

9. Clustering Observation Data

The track data is part of the network traffic flowing to and from each clustering node. Each track observation has a current position and velocity in the xy dimension. The clustering system can be expanded into a larger dimensional space, although the dimensionality is set to two currently.

The clustering nodes are interconnected using a network where multiple sensors on different platforms report the same track information but may have different accuracies. These tracks are similar representations of the same information, but are received at multiple clustering nodes at different times due to network traversal. The clustering nodes maintain information on the current clusters (active tracks) and cluster centroids. In addition to the position of the track, the system uses time stamps to determine the last time a cluster was reported.

The clustering system uses a modified algorithm similar to K-means. In traditional K-means, the number of clusters (K) is set to a specified number. We allow the number of clusters to start from zero and expand as new tracks are found.

The algorithm utilizes two tables to achieve the desired functionality. The first table, the cluster table, is comprised of the current clusters that are projections of where the track should be in the next track message. As tracks are received, they are compared to the clusters using the Manhattan distance. If the distance is less than a threshold (cluster threshold), the track message is assigned to the specified cluster. The velocity of the track message updates the projection of the cluster. This projection update allows the system to account for tracks that are not static. If the distance from the input track is greater than the threshold for all the current clusters, a new cluster is added.

The second table, the track time table, maintains two times for each cluster: the last time a message was sent about that cluster and the last time a message was received about that cluster. Since the track data is time stamped when the clustering system receives the track, the system compares the time received to the time sent. If this time is greater than the timing threshold the system will allow the message to pass and update the cluster received time in the table. However, if the time is less than the time threshold, the system updates the cluster received time and removes the message from the outgoing traffic.

10. Hardware Fabrication

The implementation on the NetFPGA platform was able achieve a clock frequency of 125 MHz. The hardware utilization of a track clustering algorithm with four parallel distance metrics is shown in Table 1. With a slice utilization of 44% the number of distance metrics can be increased. This increase in parallel distance calculation reduces the latency to compare incoming tracks to all known clusters.

Resources	**XC2VP50 Utilization**	**Utilization Percentage**
Slices	10533 out of 23616	44%
4-input LUTS	14318 out of 47232	30%
Flip Flops	12958 out of 47232	27%
Block RAMs	82 out of 232	35%
External IOBs	353 out of 692	51%

Table 1. Device utilization for XC2VP50 Hardware Track Clustering with four concepts

Given our implementation with four parallel distance metrics with 100 total tracks in the cluster table at any one time, we can estimate the total time for clustering an incoming track. The parallel distances can be calculated in four cycles. The total number of cycles required to produce all 100 distances would be 100. Since the operation of the clustering circuits are pipelined, the accept module would only require three cycles for determining the correct action to perform. The update module would require three cycles to perform an update or creation of a new cluster. The time table compare takes three cycles. Add in an additional four cycles for header processing and we have a total of 113 cycles. Since we are running the hardware at 125 Mhz, the total time required for distance calculations, assignment determination, and updates requires 0.904 μs for each incoming track. This gives us an approximate throughput of 1.1 million packets per second.

11. Conclusions

The results show that the real time hardware using the K-mean algorithm for clustering can accurately locate redundant track data for aggregation, identify new tracks, and select the critical tracks to be forwarded to other distributed sensor nodes for fusing. Improvements in the information quality and latency for distributed track fusion were demonstrated using advanced clustering.

Track data distributed using multicast is shown to generate 2.3 Gigabytes of traffic for large scale (75 nodes) fusion. Experiments revealed state-of-the-art CPUs could not handle bursts of packets received which resulted in loss of packets due to network buffer overflow. The clustering gateway node was shown to provide an 18 to 48x reduction in bandwidth through eliminating redundant data. The results were tested in a distributed environment called Emulab that used a software version of the clustering algorithms. State-of-the-art Emulab CPUs could not keep up with the track data resulting in network buffer overflow and loss of packets. This illustrates the need for our special processing solution using FPGA technology.

Real time clustering is implemented in the network layer (OSI layer 3) to reduce the bandwidth intelligently while maintaining high information content. Our pipelined hardware design calculates four parallel distance metrics for 100 total tracks in 100 cycles. The total time required for distance calculations, assignment determination, and updates is only 0.904 μs for each incoming track. The hardware solution is prototyped using the Stanford NetFPGA in the Boeing Center for Intelligent Networked Systems (CINS) lab.

Our real time solution preserves Layer 7 resources and decreases latencies. The ability to add real time hardware in the network layer improves MTT performance in bandwidth limited environments ultimately preserving legacy avionic network resources.

12. Future Work

Novel methods to illustrate the increased value of information using clustering over normal queuing methods was constructed based on spatial methods. Our use of clustering algorithms was fairly limited with K-means however there are plans to implement solutions based on N-means algorithms in the future. We realize selection of the correct clustering thresholds must be dynamic based on the separation of targets and is highly dependent on accurately projecting the centroid. The relationship between our temporal and spacial clustering algorithms must be integrated into a single solution. We plan to develop dynamic clustering thresholds for more accurate prediction of target paths while selectively reducing the distribution bandwidth. Real time assessment of the information value allows us to dynamically adjust the thresholds for preserving the key observations. We theorize that changing the update rates proportionally with the target velocity will provide improved information value. We also plan to add a weighting factor to take into account proximity, sensor accuracy and threat level will provide a more intelligent solution. Interconnecting the time increment used in the projection of the clustering centroid with the temporal target rate will simplify the algorithms and hardware.

Reliable transfer of data using wireless is available in several avionic systems and will be evaluated as a separate test case. Latency will definitely increase if all dropped packets are transferred as required by reliable transport.

Information content is improved by identifying key target characteristics such as turning ratio, threat level and multi-target separation. Increasing information content while decreasing bandwidth is the goal of the Boeing Intelligent Gateway (BIG) being developed. BIG is an intelligent gateway that uses a highly parallel state machine to implement a set of distributed services such as intelligent data association for improving the quality of information with reduced bandwidth.

REFERENCES

[1] S. S. Blackman, "Multiple-target tracking with radar applications," in *Artech House*, 1986, pp. 357–395.

[2] P. K. B.-S. Y. Chummun M. R., Kirubarajan T., "Fast data association using multidimensional assignment with clustering," in *IEEE trans. On Aerospace and Electronic Systems*, Vol. 37, No 3, 2001, pp. 898–912.

[3] S. T. K. P. Tchamova A., Dezert J., "Target tracking with generalized data assocoation based on the general dsm rule of combinaton," in *Proceedings of Fusion 2004*, Stockholm, Sweden.

[4] M. Looks, A. Levine, G. A. Covington, R. P. Loui, J. W. Lockwood, and Y. H. Cho, "Streaming hierarchical clustering for concept mining," in *Aerospace Conference (AERO)*, Mar. 2007, pp. 1–12.

[5] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. John Wiley and Sons, Mar. 1973.

[6] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of k- means clustering on reconfigurable hardware," in *FPGA*, 2001, pp. 103–110. [Online]. Available: citeseer.ist.psu.edu/estlick01algorithmic.html

[7] M. Leeser, J. Theiler, M. Estlick, N. Kitaryeva, and J. Szymanski, "Effect of data truncation in an implementation of pixel clustering on a custom computing machine," 2000. [Online]. Available: citeseer.ist.psu.edu/leeser00effect.html

[8] G. A. Covington, C. L. Comstock, A. A. Levine, J. W. Lockwood, and Y. H. Cho, "High speed document clustering in reconifigurable hardware," in *16th Annual Conference on Field Programmable Logic and Applications (FPL)*, Madrid, Spain, Aug. 2006, pp. 411–417.

[9] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," 2001. [Online]. Available: citeseer.ist.psu.edu/ng01spectral.html

[10] C. E. SHANNON, "Communication in the presence of noise," in *PROCEEDINGS OF THE IEEE, VOL. 86, NO. 2*, 1998, pp. 447–457.

[11] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "Netfpga - an open platform for gigabit-rate network switching and routing," in *International Conference on Microelectronic Systems Education*, 2007.

[12] J. Luo, J. Pettit, M. Casado, J. Lockwood, and N. McKeown, "Prototyping fast, simple, secure switches for ethane," in *Hot Interconnects*, Stanford, CA, Aug. 2007.

[13] N. Dukkipati, G. Gibb, N. McKeown, and J. Zhu, "Building a rcp (rate control protocol) test network," in *Hot Interconnects*, Stanford, CA, Aug. 2007, pp. 91–98.

BIOGRAPHY

John Meier is a Boeing Technical Fellow in the Network Centric Thrust at Phantom Works, NCO Thrust. He has over 27 years of professional experience in avionic technology development specifically working in the area of intelligent networking, reconfigurable computing architectures and wireless network management. At Boeing, he is currently involved with several key technical activities including a major project on edge computing and intelligent distributed system management. Mr. Meier earned his BS from Southern Illinois University - Carbondale (SIU-C), MS University of Missouri - Rolla (UMR), and currently working on his PhD degrees from the Department Computer Science Engineering (CSE) at the Washington University in St. Louis. He is a member of IEEE and Tau Beta Pi.

Todd Sproull is a Doctoral Candidate pursuing his DSc in Computer Engineering at Washington University in Saint Louis. He is a Research Assistant and member of the Reconfigurable Network Group (RNG) at Washington University. His interests include peer-to-peer overlay networks, reconfigurable computing and network security. He has worked in the industry at Xilinx Research Labs, Network Physics, and IBM. He earned a BS degree in Electrical Engineering at Southern Illinois University in Edwardsville and an MS in Computer Engineering at Washington University. He is a member of IEEE, Tau Beta Pi and Eta Kappa Nu.

G. Adam Covington is a Research Associate of the Reconfigurable Network Group (RNG) at Washington University in St. Louis. Adam's research interests include reconfigurable systems, artificial intelligence (clustering and classification), and applications of artificial intelligence algorithms. Upon completing a Bachelor of Science degree in Computer Engineering in 2003, Adam earned his Masters of Science degree in Computer Science and Engineering from Washington University in December of 2006. He is currently visiting Stanford to implement data clustering on the NetFPGA platform and help assemble NetFPGA test systems.

John W. Lockwood designs and implements networking systems in reconfigurable hardware. During 2007, he served as a Visiting Associate Professor at Stanford University to manage the Alpha and Beta releases of the NetFPGA. Prior to working at Stanford, he led the the Reconfigurable Network Group (RNG) at Washington University. The RNG research group developed the Field programmable Port Extender (FPX) to enable rapid prototype of extensible network modules in Field Programmable Gate Array (FPGA) technology. As an Associate professor in the Department of Computer Science and Engineering at Washington University in Saint Louis. He has published over 75 full-length papers in journals and major technical conferences that describe technologies for providing extensible network services in wireless LANs and in high-speed networks. Professor Lockwood has served as the principal investigator on grants from the National Science Foundation, Xilinx, Altera, Nortel Networks, Rockwell Collins, and Boeing. He has worked in industry for AT&T Bell Laboratories, IBM, Science Applications International Corporation (SAIC), and the National Center for Supercomputing Applications (NCSA). He served as a co-founder of Global Velocity, a networking startup company focused on high-speed data security. Dr. Lockwood earned his MS, BS, and PhD degrees from the Department of Electrical and Computer Engineering at the University of Illinois. He is a member of IEEE, ACM, Tau Beta Pi, and Eta Kappa Nu.