ASAP: A Camera Sensor Network for Situation Awareness*

Junsuk Shin, Rajnish Kumar, Dushmanta Mohapatra,
Umakishore Ramachandran, and Mostafa Ammar

College of Computing, Georgia Institute of Technology, Atlanta GA, USA
{jshin,rajnish,dmpatra,rama,ammar}@cc.gatech.edu

Abstract. *Situation awareness* is an important application category in cyber-physical systems, and *distributed video-based surveillance* is a good canonical example of this application class. Such applications are interactive, dynamic, stream-based, computationally demanding, and needing real-time or near real-time guarantees. A *sense-process-actuate* control loop characterizes the behavior of this application class. ASAP is a scalable distributed architecture for a multi-modal sensor network that caters to the needs of this application class. Features of this architecture include (a) generation of *prioritization* cues that allow the infrastructure to pay *selective attention* to data streams of interest; (b) *virtual sensor* abstraction that allows easy integration of multi-modal sensing capabilities; and (c) dynamic redirection of sensor sources to distributed resources to deal with sudden burstiness in the application. In both empirical and emulated experiments, ASAP shows that it scales up to a thousand of sensor nodes (comprised of high bandwidth cameras and low bandwidth RFID readers), significantly mitigates infrastructure and cognitive overload, and reduces *false negatives* and *false positives* due to its ability to integrate multi-modal sensing.

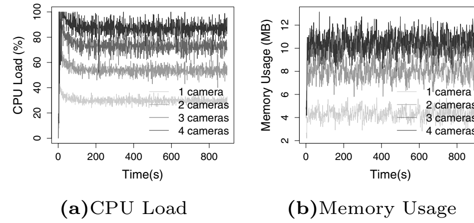
1 Introduction

Situation Awareness is both a property and an application class that deals with recognizing when *sensed data* could lead to *actionable knowledge*. However, because of a huge increase in the amount of sensed data to be handled, providing situation awareness has become a challenge. With advances in technology, it is becoming feasible to integrate sophisticated sensing, computing, and communication in a single small footprint sensor platform. This trend is enabling deployment of powerful sensors of different modalities in a cost-effective manner. While Moore's law has held true for predicting the growth of processing power, the volume of data that applications handle is growing similarly, if not faster.

There are three main challenges posed by data explosion for realizing situation awareness: overload on the infrastructure, cognitive overload on humans in the loop, and dramatic increase in false positives and false negatives in identifying threat scenarios. Consider, for example, providing situation awareness in a

* ASAP stands for "Priority Aware Situation Awareness" read backwards.

E. Tovar, P. Tsigas, and H. Fouchal (Eds.): OPODIS 2007, LNCS 4878, pp. 31–47, 2007.
© Springer-Verlag Berlin Heidelberg 2007



(a)CPU Load **(b)**Memory Usage

Fig. 1. Resource usage in a centralized camera network: cameras produce data at 5 fps with 320x240 resolution. Image processing happens on a 1.4GHz Pentium processor.

battlefield. It needs complex fusion of contextual knowledge with time-sensitive sensor data obtained from different sources to derive higher-level inferences. With an increase in the sensed data, a fighter pilot will need to take more data into account in decision-making leading to a cognitive overload and an increase in human errors (false positives and negatives). Also, to process and disseminate the sensed data, more computational and network resources are needed thus overloading the infrastructure.

The severity of infrastructure overload is more apparent for camera sensor networks because image dissemination and processing tasks are very resource intensive. Consider, for example, a simple surveillance system that does motion sensing and JPEG encoding/decoding. Figure 1 shows the processing requirements for such a system using a centralized set up: cameras produce data at 5 frames/second with a 320x240 resolution; image processing happens on a 1.4GHz Pentium processor. The results show that the above centralized setup cannot scale beyond four cameras (the CPU load is nearly 100%.) If we increase the video quality (frames/second and resolution), even a high-end computing resource will be unable to process more than a few cameras.

Clearly, scaling up to a large number of cameras (on the order of 100's or 1000's) warrants a distributed architecture. Further, to mitigate the challenges posed by the data explosion there is a need to add a *prioritize* step in the control loop for situation awareness. The ASAP architecture presented in this paper caters to the *sense-process-prioritize-actuate* control loop. Adding the prioritize step is expected to help not only in an effective use of the available resources, but also to achieve scalability and meet real-time guarantees in the data deluge.

ASAP provides features that are aimed to address the specific challenges posed by situation awareness application:

– It provides a framework for generating *priority cues* so that the system (and humans in the loop) can pay *selective attention* to specific data streams thus reducing both the infrastructure and cognitive overload.
– It consists of two *logical* networks, namely, *control* and *data*. The former generates priority cues and the latter provides processing functions (filtering and fusion) on the selected data streams. This tiered architecture enables the physical network consisting of cameras, sensors, and computational resources to be scaled up or down more easily. Further this logical separation aids in dealing with sudden burstiness in the sensed environment.

- It provides facilities for dynamic redirection of control and data streams to different computational resources based on the burstiness of the sensed environment.
- It provides an abstraction, *virtual sensor*, that allows sensors to operate multi-modally to reduce the ill effects of false positives and false negatives.
- It integrates hand-held devices (such as iPAQs) to enable flexible delivery of alerts and information digests.

The unique contributions of our work are as follows: (a) a systematic approach to help prioritize data streams, (b) a software architecture that ensures scalability and dynamic resource allocation, and (c) multi-model sensing cues to reduce false positives and false negatives.

The rest of the paper is organized as follows. Section 2 explores situation awareness applications to understand their requirements. Section 3 explains the ASAP architecture and its prioritization strategies. The implementation and evaluation of ASAP platform are presented in Sections 4 and 5, respectively. Related work is discussed in Section 6. Section 7 concludes the paper.

2 Understanding ASAP Requirements

2.1 Example Application: Video Based Surveillance

A video-based surveillance system is an attractive solution for threat identification and reduction. Cameras are deployed in a distributed fashion; the images from the cameras are filtered in some application-specific manner, and are fused together in a form that makes it easy for an end user (human or some program) to monitor the area. The compute intensive part may analyze multiple camera feeds from a region to extract higher-level information such as “motion”, “presence or absence of a human face”, or “presence or absence of any kind of suspicious activity”. Security personnel can specify a set of security policies that must be adhered to, e.g. “only specified people are allowed to enter a particular area”, and the system must continuously ensure that an alert is generated whenever any breach happens. Similarly, security personnel can do a search on all available camera streams for an event of interest, e.g. “show me the camera feed where there is a gas leak”. To support the above two ways of deriving actionable knowledge, the extracted information from camera streams, e.g. motion or number of faces etc., may be the meta-data of importance for information prioritization. With a large number of surveillance cameras (e.g. 3K in New York [1] and 400K in London [2]), it becomes a more interesting issue.

2.2 Application Requirements

Applications such as video-based surveillance are capable of stressing the available computation and communication infrastructures to their limits. Fusion applications, as we refer to such applications in this paper have the many common needs that should be supported by the underlying ASAP platform:

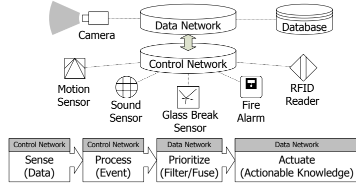


Fig. 2. Functional view of ASAP

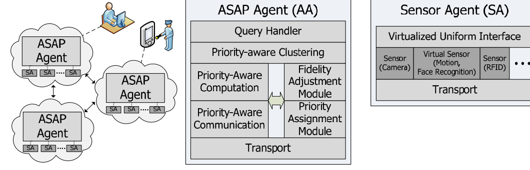


Fig. 3. ASAP Software Architecture

1. High Scalability: The system should scale to large number of sensor streams and user queries. This necessarily means that the system should be designed to reduce infrastructure overload, cognitive overload, and false positives and false negatives. False positives refer to the actionable knowledge triggers generated by the system that turns out to be not really a security threat or an event of interest. False negatives refer to the security threat situations or interesting events that are missed by the system.
2. Query vs Policy-based interface: Situation awareness applications need to support both query- and policy-based user interfaces. A query-based interface will allow users to search streams of interest based on tags or information associated with the streams. On the other hand, a policy-based interface will allow users to specify conditions to monitor and to generate alerts based on the conditions. A set of policies can be specified by a security administrator to proactively monitor an area. A platform for situation awareness applications should provide both the query-based and policy-based mechanisms.
3. Heterogeneity and Extensibility: With advances in sensing technologies, it has become possible to deploy different types of sensors on a large scale to derive actionable knowledge. Further, since a single type of sensor may not be sufficient to provide accurate situational knowledge, there is a need to use different types of sensors to increase the accuracy of event detection. There is also a need to use different types of sensors, because a single sensing modality is often not sufficient to provide accurate situational knowledge. For use in diverse application scenarios, it is imperative that ASAP accommodate heterogeneity of sensing, while being flexible and extensible.

3 Architecture

Figure 2 shows the logical organization of the ASAP architecture into control and data network. The control network deals with low-level sensor specific processing to derive priority cues. These cues in turn are used by the data network to prioritize the streams and carry out further processing such as filtering and fusion of streams. It should be emphasized that this logical separation is simply a convenient vehicle to partition the functionalities of the ASAP architecture. The two networks are in fact overlaid on the same physical network and share the computational and sensing resources. For example, low bitrate sensing such as an RFID tag or a fire alarm are part of the control network. However, a high bitrate camera sensor while serving the video stream for the data network may also be used by the control network for discerning motion.

Figure 3 shows the software architecture of ASAP: it is a peer-to-peer network of ASAP agents (AA) that execute on independent nodes of the distributed system. The software organization in each node consists of two parts: *ASAP Agent (AA)* and *Sensor Agent (SA)*. There is one sensor agent per sensor, and a collection of sensor agents are assigned dynamically to an ASAP agent.

3.1 Sensor Agent

SA provides a *virtual sensor* abstraction that provides a uniform interface for incorporating heterogeneous sensing devices as well as to support multi-modal sensing in an extensible manner. This abstraction allows new sensor types to be added without requiring any change of the ASAP agent (AA). There is a potential danger in such a virtualization that some specific capability of a sensor may get masked from full utilization. To avoid such semantic loss, we have designed a minimal interface that serves the needs of situation awareness applications.

The virtual sensor abstraction allows the same physical sensor to be used for providing multiple sensing services. For example, a camera can serve not only as a video data stream, but also as a motion or a face detection sensor. Similarly, an SA may even combine multiple physical sensors to provide a multi-modal sensing capability. Once these different sensing modalities are registered with ASAP agents, they are displayed as a list of available features that users can select to construct a query for ASAP platform. ASAP platform uses these features as control cues for prioritization (see Section 3.2).

3.2 ASAP Agent

As shown in Figure 3, an AA is associated with a set of SAs. The association is dynamic, and is engineered at runtime in a peer-to-peer fashion among the AAs. The components of AA are shown in Figure 3.

Query Interface

ASAP agent provides a simple query interface with SQL-like syntax. Clients can pose an SQL query using control cues as attributes. Different cues can be combined using "AND" and "OR" operators to create multi-modal sensing queries. Here are some example queries which are self evident as to their intent: [1]

```
1) SELECT images FROM zone("Gate13")  WHERE RFIDTag = 'James'
2) SELECT images FROM zone("any")  WHERE FaceRecognition = 'Alice'
3) SELECT COUNT(Object) FROM zone("Concourse B")
```

False Positives and Negatives

Figure 2 shows that *sensed data* leads to *events*, which when filtered and fused ultimately leads to *actionable knowledge*. Unfortunately, individual sensors may often be unreliable due to environmental conditions (e.g., poor lighting conditions near a camera). Thus it may not always be possible to have high confidence

[1] It should be understood that the above queries are just a few examples. The interface is extensible to support different types of sensors, as well as, dispense both streams and digests of streams as the query output.

in the sensed data; consequently there is a danger that the system may experience high levels of false negatives and false positives. It is generally recognized that multi-modal sensors would help reduce the ill effects of false positives and negatives. The virtual sensor abstraction of ASAP allows multiple sensors to be fused together and registered as a new sensor. Unlike multi-feature fusion (*a la* face recognizer) where features are derived from the same (possibly noisy) image, multi-sensor fusion uses different sensing modalities. ASAP exploits a quorum system to make a decision. Even though a majority vote is implemented at the present time, AA may assign different weights to the different sensors commensurate with the error rates of the sensors to make the voting more accurate.

Prioritization Strategies

ASAP needs to continuously extract prioritization cues from all the cameras and other sensors (control network), and disseminate the selected camera streams (data network) to interested clients. ASAP extracts information from a sensor stream by invoking the corresponding SA. Since there may be many SAs registered at any time, invoking all SAs may be very compute intensive. ASAP needs to prioritize the invocations of SAs to scale well with the number of sensors. This leads to the need for *priority-aware computation* in the control network. Once a set of SAs that are relevant to client queries are identified, the corresponding camera feeds need to be disseminated to the clients. If the bandwidth required to disseminate all streams exceed the available bandwidth near the clients, network will end up dropping packets. This leads to the need for *priority-aware communication* in the data network. Based on these needs, the prioritization strategies employed by ASAP can be grouped into the following categories: Priority-aware computation and priority-aware communication.

Priority-aware Computation. The challenge is dynamically determining a set of SAs among all available SAs that need to be invoked such that overall value of the derived actionable knowledge (benefit for the application) is maximized. We use the term *Measure of Effectiveness (MOE)* to denote this overall benefit. ASAP currently uses a simple MOE based on clients' priorities.

The priority of an SA should reflect the amount of possibly "new" information the SA output may have and its importance to the query in progress. Therefore, the priority value is dynamic, and it depends on multiple factors, including the application requirements, and the information already available from other SAs. In its simplest form, priority assignment can be derived from the priority of the queries themselves. For instance, given two queries from an application, if the first query is more important than the second one, the SAs relevant to the first query will have higher priority compared to the SAs corresponding to the second query. More importantly, computations do not need to be initiated at all of SAs since (1) such information extracted from sensed data may not be required by any AA, and (2) unnecessary computation can degrade overall system performance. "WHERE" clause in SQL-like query is used to activate a specific sensing task. If multiple WHERE conditions exist, the lowest computation-intensive task is initiated first that activates the next task in turn. While it has a trade-off between latency and overhead, ASAP uses this for the sake of scalability.

(a) Axis 207MW (b) RFID Antenna (c) ASAP client on iPAQ

Fig. 4. Testbed building blocks

Priority-aware Communication. The challenge is designing prioritization techniques for communication on data network such that application specific MOE can be maximized. Questions to be explored here include: how to assign priorities to different data streams and how to adjust their *spatial* or *temporal* fidelities that maximizes the MOE?

In general, the control network packets are given higher priority than data network packets. Since the control network packets are typically much smaller than the data network packets, supporting a cluster of SAs with each AA does not overload the communication infrastructure.

4 Implementation

We have built an ASAP testbed with network cameras and RFID readers for object tracking based on RFID tags and motion detection. In implementing ASAP, we had three important goals: (1) platform neutrality for the "box" that hosts the AA and SA, (2) ability to support a variety of sensors seamlessly (for e.g., network cameras as well as USB cameras), and (3) extensibility to support a wide range of handheld devices including iPAQs and cellphones. Consequent to these implementation goals, we chose Java as the programming language for realizing the ASAP architecture. Java also provides Java Media Framework (JMF) API [3] that supports USB cameras on many platforms.

Table 1. Axis 207MW specifications

Specifications	
Video Compression	Motion JPEG, MPEG-4
Resolutions	15 resolutions up to 1280x1024
Frame Rate	Up to 14 fps up to 1280x720, Up to 12 fps in 1280x1024
Wireless interface	IEEE 802.11g 6-54 Mbps, IEEE 802.11b 1-11 Mbps

Figure 4 shows the building blocks of our testbed: a network camera, Axis 207MW from Axis Communication [4] and RFID antenna from Alien Technology [5]. The key specifications of the network camera are given in Table 1. Considering iPAQ[2] performance, we decided to use motion JPEG with 320x240

[2] At the time of writing, our implementation uses an iPAQ and/or a desktop for the GUI client. We plan to extend the implementation to include a cellphone through web service in the near future.

resolution, 5 fps, and 40 compression. Higher compression value (0–100) corresponds to lower quality and smaller image size. A JPEG frame requires 8–14 KBytes depending on the image content.

ASAP implementation consists of 3 main components: GUI Client, Sensor Agent, and ASAP Agent. GUI client has a simple interface to send a query. In a simple query, a user can select from drop-down lists for features such as RFID tag and motion detection, which tag needs to be tracked, and how many output streams he/she would like to receive. For a more complicated query such as tracking based on multiple tags and/or multiple features, SQL query is used. Then, the query is represented as an XML document and sent to the nearest ASAP Agent (either through the wired network or wirelessly depending on the connectivity of the client to the ASAP infrastructure). The client uses a *name server* to discover a nearby ASAP agent. In our current implementation, a text file at a well-known location serves the purpose of a name server. While there could be a debate about a scalability issue in this kind of naming service, the deployment of camera surveillance system is usually static, and AA keeps caches of topology information. Even in the case of dynamic deployment, ASAP can easily integrate DNS-like service which is not the focus of this work.

4.1 Sensor Agent

Sensor Agent needs to provide as simple as possible interface to alleviate the development of different types of sensors. This component requires frequent changes and updates due to the changes in detection algorithms or addition of sensors. The development of SA consists of 3 steps. The first step is the development of sensor functionality. It can be either physical sensor functions such as RFID tag reading or virtual sensor like motion detection or face recognition. The second step is the registration of sensor through a uniform interface. A control message handler is the last step. ASAP supports rich APIs to ease the second and third steps, and an application programmer can focus only on the first step. By having a tiered network architecture, Sensor Agent and ASAP Agent are functionally less dependent upon each other. This makes the ASAP software easy to understand, maintain, and extend with different sensors types.

The virtual sensor abstraction serves to make implementing new sensor functionality a cinch in ASAP. For e.g., given a camera, if the developer decides to implement two different functionalities (say, face recognition and motion detection) using the camera, then she would register each as a distinct sensor agent (SA) with the ASAP agent. This componentization of SAs allows modular extension to the overall architecture without any central control ensuring the scalability of the architecture.

For the testbed, we implemented camera, motion, and RFID sensors. In the case of camera sensors, ASAP supports USB cameras and Axis network cameras. JMF is used to support USB cameras, and it supports Windows, Linux, and Solaris. Axis supports HTTP API called VAPIX API. By sending HTTP request, camera features can be controlled, and images or streams can be retrieved. By sending a request, `http://[address]/axis-cgi/mjpg/video.cgi`, a camera is turned on and starts sending motion JPEG. With JMF, the camera URL is

represented as following; `vfw://0` on Windows platform or `v4l://1` on Linux platform. The last digit starts from 0 to the number of USB cameras attached exclusively. ASAP provides APIs for uniformly accessing cameras independent of their type. It implicitly figures out the camera type from the URL once again reducing the programming burden on the developer.

The data from the Sensor Agent may be directed dynamically to either the control network or the data network at the behest of the ASAP agent. The command from AA to SA specifies start/stop as well as periodicity (for periodic sensors, see Section 3.1). Alert sensors simply send a binary output. For instance, the RFID reader responds yes/no to a query for tracking a specific tag. It is possible for the user to control the amount of communication generated by an SA using the query interface. For example, if the user sets a threshold of motion, then this will be communicated by AA in a command to the SA. Upon receiving such a command, the associated SA for motion detection needs to send an alert only when the level of motion crosses the threshold specified by the AA. Even for a periodic stream (such as a video stream) communication optimization is possible from the query interface using the WHERE clause.

We implemented Java motion detection based on open source and RFID sensor using Alien Technology APIs. Since the ASAP agent is responsible for all command decisions regarding the granularity of operation of the SAs, it was easy to implement a variety of sensors (including multi-modal ones). Our experience validates our claim regarding the utility of the virtual sensor abstraction.

4.2 ASAP Agent

ASAP Agent (AA) is the core component of ASAP system. Not only does it handle multiple clients and Sensor Agents, but also communicates with other AAs in a distributed manner. ASAP Agent works as a delegate of a client. Since a client does not have global knowledge (e.g. how many cameras or RFID readers are deployed), it picks an ASAP Agent, and sends queries. AA should meet the following requirements: (1) efficient handling of multiple client queries, (2) efficient management of the control and data networks, and (3) dynamic load balancing via assignment of SAs to AAs.

Query Handler Module

Query handler module receives queries from multiple clients. An ASAP Agent that receives a query, interprets it, and decides on which peer AAs to activate on the control network. For example, a security guard may issue the following query to find where in Gate 11 "Ellice" is and request to receive one camera stream if she is in Gate 11.

```
SELECT images FROM zone('Gate 11') WHERE RFIDTag = 'Ellice'
```

As we mentioned in Section 3.2, each AA handles a cluster of SAs. There is no global knowledge of SA to AA association. A given AA knows the attributes of its peers (for e.g., which AA is responsible for SAs in Gate 11). Thus, upon receiving this query, the AA will forward the query to the appropriate AA using

the control network. Upon receiving the forwarded query, the AA for Gate 11 will issue commands to its local SAs (if need be) to satisfy the query. If the AA already has the status of the SAs associated with it, then it can optimize by responding to the query without having to issue commands to its SAs.

Priority Assignment Module

The function of the priority assignment module is three-fold: 1) control message management, 2) relative priority assignment, and 3) data network management. Each of these functions is implemented by separate sub-modules.

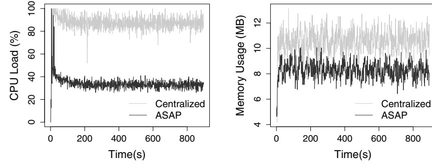
Control message management sub-module maintains a *client request map.* Some of these requests may be from local clients, while others may be forwarded from peer AAs. If the request needs local resources, then it hands it to the relative priority assignment sub-module. If it requires remote resources, then it is forwarded as a control message to the peer AA. Communication is saved when a new client request can be satisfied by a pending remote request to the peer.

Relative priority assignment sub-module assigns priority values to various data streams. The priorities are assigned in the range {high,medium,low}. ASAP uses only three values for priority assignment due to a simple and efficient priority queue management and different streams are assigned to these queues. All streams belonging to the queue for one priority level are treated in the same way. This coarse grained priority assignment suits very well to ASAP. While more fine grained priority assignments are possible, they increase the complexity of implementation and overhead in queue managements.

The priority assignment happens over a series of steps and can take place in two different ways. The first is a top-down approach where a client query has a priority associated with it. In this case streams satisfying a query are assigned the priority associated with the query. If one stream satisfies the queries from multiple clients, the highest priority value among the queries is assigned to it. The accumulated or average priority among the queries can lead to a priority inversion. After a priority is assigned, streams are split into 3 groups for queues of different priority levels.

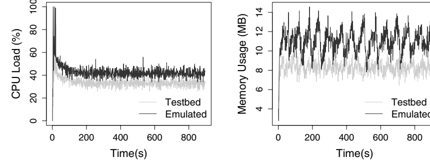
With a bottom-up approach, ASAP assigns a priority to a stream. Since there is no correlation among streams that meet distinct queries, this assignment occurs when a client requests for more than one stream or the conditions in a query are satisfied by multiple streams. In this situation AA assigns priority values ranging in {high,medium,low} to these streams. For instance, when a client requests a highest motion detection limited by 3 streams, a stream with the highest commotion will have {high} priority. As in a top-down approach, if a stream is requested by multiple clients, ASAP chooses the highest priority.

Data network management sub-module sends control messages to SAs indicating the data network to be turned on or off. The control messages also contain the priority assigned to the data network. The same scheme of control message management sub-module is used to manage request map of streams, and both control and data network are optimized to reduce redundant transmission.



(a)CPU Load **(b)**Memory Usage

Fig. 5. Resource usage (Centralized vs. ASAP): A single object tracking system based on RFID tag. Cameras produce data (320x240 5 fps).



(a)CPU Load **(b)**Memory Usage

Fig. 6. Testbed vs. Emulated: A single object tracking system based on RFID tag with 4 cameras, 4 motion sensors, and 4 RFID readers

5 Evaluation

Before we go into the details of our scalability results, it is worth looking at how ASAP handles the concern that was raised in the introduction section. With a simple setup of 4 cameras, 4 motion sensors, 4 RFID readers, and a single client, we showed in Figure 1 that the CPU usage on a typical desktop system is close to 100%. Figure 5 shows the same setup using the prioritization strategy of ASAP and compares it with the 4-camera result from Figure 1. In this setup, ASAP uses a specific RFID tag as a control cue to decide the camera stream to be processed. As can be seen, use of control cues to select the camera stream results in a 60% improvement in CPU load. This establishes a baseline of expectation for performance improvement with the prioritization strategies of ASAP, and the promise ASAP offers for reducing the infrastructure overload. In the following subsections, we detail the experimental setup and the performance results of our scalability studies.

5.1 Experimental Setup

Since our current testbed has only a limited number of real cameras and RFID readers, the testbed is not enough for a large scale evaluation. Therefore, we developed emulated sensors support using the uniform virtual sensor interface discussed in Section 3. Due to the virtual sensor abstraction, an *ASAP Agent* does not distinguish whether data comes from an emulated sensor or a real sensor. The emulated camera sends JPEG images at a rate requested by a client. The emulated RFID reader sends tag detection event based on an event file, where different event files mimic different object movement scenarios.

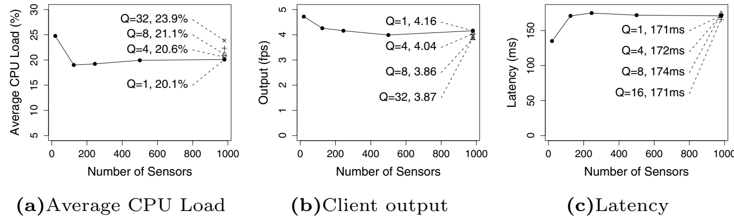
To understand the impact of using emulated sensors on our results, we performed an experiment to compare the resource usage of emulated sensors with that of real sensors. Figure 6 shows the comparison. This experiment uses a network of 4 camera sensors, 4 RFID, and 4 motion detection, for a single object tracking. Because emulated sensors generate images and read from event files, they consume more CPU and memory resources than real sensors. However, the results show that the emulated setup is close enough to the real testbed thus validating our scalability studies with emulated sensors.

Table 2. Workload Parameters

Parameter	Configuration
Number of SAs	20, 125, 245, 500, 980
Image Format	M-JPEG 320x240 @ 5 fps
Number of Queries	1, 4, 8, 16, 32
Multi-Modality	1, 2, 3-Modality

Table 3. Cluster Specification

CPU	Dual Intel Xeon 3.2 GHz
Memory	6GB
Network	Gigabit Ethernet
Number of Nodes	53
OS	Linux (Kernel 2.6.9)



(a) Average CPU Load **(b)** Client output **(c)** Latency

Fig. 7. Scalability results: The solid line represents the effect of the number of SAs on different scalability metrics for a single query. The dotted lines point to the scalability results when the number of queries is varied from 1 ($Q = 1$) to 32 ($Q = 32$) for a setup with 980 SAs.

Workload

For the following experiments, *workload* used is as follows. An area is assumed to be made of a set of cells, organized as a grid. Objects start from a randomly selected cell, wait for a predefined time, and move to a neighbor cell. The number of objects, i.e. the number of RFID tags, the grid size, and the object wait time are workload parameters.

Table 2 summarizes the parameters used in our experiments. The number of SAs is varied from 20 to 980. An ASAP agent is assigned for every 20 SAs. For e.g., for a setup with 20 SAs, there will be one ASAP agent, and for a setup with 125 SAs, there will be 6 ASAP agents. Each ASAP agent runs on a distinct node of a cluster (see Table 3) of dual Intel Xeon 3.2 GHz processors with 6GB of RAM running Linux. A fixed experiment duration (15 minutes) is used through all performance evaluations. Other experimental parameters are explained below in the context of the specific experiments.

5.2 Scalability, Query Handling, and Latency

Figure 7 shows scalability results for tracking a single object when the number of SAs is increased. This corresponds to an application scenario wherein the movement of a suspicious individual carrying a boarding pass with a specific RFID is tracked in an airport. To handle the increase in the number of cameras and other sensors, more ASAP Agents are added (with 20:1 ratio between SAs and ASAP

agent). Figure 7(a) shows the average CPU load over all the ASAP agents for a particular configuration. On each node, processing cycles are used for ASAP agent, SAs, and performance monitoring. With a single query, only one ASAP agent in the entire system has to do the heavy lifting. While there is processing cycles devoted to SAs and monitoring in each node of the distributed system despite the fact that there is just a single query, the prioritization architecture ensures that the CPU load due to the SAs on each node is pretty minimal. Since the Y-axis is the CPU load averaged over all the nodes, there is an initial drop in the average CPU load (from 25% to 19%) and then it remains the same at about 20%. The fact that the average CPU load remains the same despite the size of the deployment (with 980 sensors we use 49 nodes of the cluster) is confirmation of the scalability of the ASAP architecture. As a side note, the CPU load on each node due to performance monitoring is 7.5%.

For the maximum workload configuration of 980 SAs, Figure 7(a) also shows how ASAP scales with varying number of queries (clients). The multiple query experiment assumes the queries are independent of one other and are emanating from distinct clients. This corresponds to an application scenario wherein the movement of multiple suspicious individuals carrying boarding passes tagged with distinct RFIDs are tracked in an airport by different security personnel. Increasing the number of queries increases the average CPU load, but at a very low rate. For example, when the number of queries increases from one to 32, the average CPU usage per node increases only by 4%.

Figure 7(b) shows the scalability of ASAP for delivering output (video streams) to multiple clients. The workload (Table 2) fixes the camera data generation at 5 fps. Ideally, we would like to see this as the delivered frame rate to the clients. With the single node vanilla system that we discussed in the introduction (Section 1), we observed an output delivery rate of 3 fps (a companion measurement to Figure 1). As can be seen in Figure 7(b), the average output delivery rate is over 4.26 fps over the range of SAs we experimented with. The frame rate degrades gracefully as the size of the system is scaled up (along the x-axis). Even when the number of queries increase, the frames per second degrades gracefully, for e.g., with 32 queries, ASAP delivers (Figure 7(b)) on an average 3.87 fps over the range of SAs we experimented with.

Figure 7(c) shows the end-to-end latency measurement as the system size is scaled up (along the x-axis). The measured time is the elapsed time between receiving a frame at the SA associated with a camera to the time it is delivered to a client. This latency is 135 ms with a single AA. As the system is scaled up the source SA and the destination client may be associated with different nodes (i.e., different AAs as shown in Figure 3) requiring a forwarding of the data stream. However, as can be seen from Figure 7(c), the forwarding only slightly increases the end-to-end latency as the system is scaled up. On an average the latency is 170 ms over the range of SAs we experimented with. Similarly, the latency is not affected tremendously with the number of queries. In fact with 16 queries, there is even a reduction in the latency which may be attributed to perhaps the source SA and the destination client being collocated at an AA more often than not (thus reducing the forwarding messages incurred).

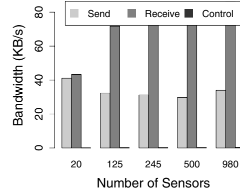


Fig. 8. Bandwidth usage: The number of SAs is varied from 20 to 980 with a single query

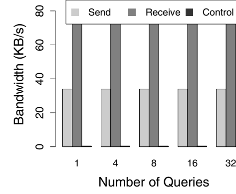


Fig. 9. Average Bandwidth: the number of queries is varied from 1 to 32 with the largest configuration of 980 SAs

5.3 Network Bandwidth

Another important resource to consider as we scale up the system size is the network bandwidth. Similar to the CPU load experiments in Section 5.2, we wish to understand how the network bandwidth requirement changes with increasing system size and increasing number of queries. As a point of reference, the observed raw bandwidth requirement per M-JPEG stream (from the Axis camera) is 40 KBytes/sec to 60 KBytes/sec (at 5 fps) depending on the contents of the image frame. There are three sources of demand on network bandwidth in the ASAP architecture: an AA *receiving* data streams from sensors via associated SAs (including forwarding to peer AAs), an AA *sending* data streams to clients, and AAs communicating control information with another. The first two are demands placed on the physical network by the data streams and the third by the control streams. Figure 8 shows the network bandwidth used for a single query when the system is scaled up. The *send* bandwidth remains roughly the same and tracks the frames/sec delivered to the clients in Figure 7(b). The *receive* bandwidth roughly doubles beyond one node and represents the data forwarding between AAs. However, it stays pretty much the same independent of the system size showing the scalability of the ASAP architecture. The reason for the doubling is due to the fact that the workload assumes random positioning of the object being tracked (over the 15 minute window of experimentation time); thus the larger the network the more the chance of data being forwarded between AAs. However, there is at most one hop of forwarding due to the peer-to-peer nature of the AA arrangement in the ASAP architecture. The control traffic (small black line which is almost invisible) is just 2% of the total bandwidth usage independent of the system size due to the variety of optimizations that we discussed in Section 4.2.

Figure 9 shows the average bandwidth usage for increasing number of queries. This experiment uses the maximum workload configuration of 980 SAs (Table 2). As with the CPU load experiments, each client query is tracking a different object. We do have results when all the clients request the same object to be tracked but have not presented them in this paper for space considerations. As may be expected such clustering of requests results in reducing the network requirements (specifically the receive and control traffic bandwidths are reduced).

5.4 Other Results

We have also conducted experiments to test false positives and false negatives. ASAP uses a dynamic voting mechanism to reduce the ill effects of false positive and false negatives. These mechanisms result in reduction in the range of 18%-64% for false-positives and -negatives. Due to space limitations, we do not present these experiments and results.

6 Related Works

There have been other interesting researches on architecture for camera sensor networks, which have motivated ASAP's two-tier approach of control and data networks. IrisNet [6] provides an architecture for a worldwide sensor web. It also shares commonalities with ASAP such as agent-based approach, two-tier network, and heterogeneous sensors support. The major difference lies in the main goal and target applications. IrisNet focuses on distributed database techniques for gathering sensor data and querying the collected data. The result of query in IrisNet is a digest of information culled from the collected data stored in the distributed databases. The focus of ASAP is to prioritize and prune data collection in an application-specific manner to deal with the data explosion so that irrelevant data is neither collected, nor processed, nor stored. Further, ASAP provides a continuous stream of information as the query result (which may be real-time data streams or digests of processing such real-time streams) satisfying the query constraints. The techniques for storing and retrieving data and maintaining consistency of the distributed databases, a forte of IrisNet project, is a nice complement to our work.

SensEye [7] is an architecture for multi-tier camera sensor network. SensEye uses three tier networks, and sense-actuate control loop exists from lowest tier (with low resolution cameras) to the highest (with higher resolution cameras). While SensEye focuses on the reduction of power consumption, having 3-tiered network can increase the complexity of software architecture. Tenet [8] also uses a tiered architecture. However, the focus of Tenet lies on the architecture support for simplifying application development and concurrent application, while ASAP focuses on how to query camera sensors in a scalable manner.

A natural way to think about managing resources in situation awareness applications is to leverage the application dataflow. For example, RF^2ID [9] is a middleware that uses the flow of tagged objects to create a group of nodes that can process the data generated from those objects. Similarly, the concept of flow is used to support QoS-aware routing [10]. In such situations, location-based correlation [11] can facilitate the desirable clustering of the nodes. Also, techniques from on-demand clustering [12] can be used to further reduce the amount of communication required to do reclustering in a dynamic environment. Finally, apart from using just location-based attributes, other attributes can also be used to achieve an application-aware clustering [13].

Supporting QoS in network layers has been an active area of research because of the growing popularity of cell phones and multimedia services. QoS-aware

medium access control protocols have been proposed to handle different techniques to match with different data types, for e.g., IEEE 801.11e handles four different data classes for better QoS support [14]. To handle real-time requirements for QoS-aware routing, common techniques used are rate-control techniques at data sources and route adjustments by finding least-cost and delay-constrained links [15]. Similarly, rate control and congestion control mechanisms have been used to provide QoS-aware transport protocols [16]. Our application-layer approach to do the priority assignment is complementary to and fits on top of the above network layer techniques.

7 Conclusions

Situation awareness is an important application category in cyber-physical systems. Video-based surveillance is an example of this category of applications. There is an explosion in the amount of data that has to be dealt with in such applications, especially as the data sources scale up.

ASAP is a distributed systems architecture for camera based sensor networks that deals with this data deluge. The unique feature of ASAP is a systematic approach to prioritizing the data streams, and the subsequent processing of these streams using cues derived from a variety of sensory sources. Further, the peer-to-peer nature of the ASAP architecture ensures scalability to a large number of camera sources, and for the dynamic allocation of computational resources to hot spots in the application. Lastly, the system provides a systematic way for reducing false positives and false negatives using multi- modal sensing.

Acknowledgements

The work has been funded in part by an NSF ITR grant CCR-01-21638, NSF NMI grant CCR-03-30639, NSF CPA grant CCR-05-41079, and the Georgia Tech Broadband Institute. The equipment used in the experimental studies is funded in part by an NSF Research Infrastructure award EIA-99-72872, and Intel Corp. We thank the members of the Embedded Pervasive Lab at Georgia Tech (`http://wiki.cc.gatech.edu/epl/`) for their helpful feedback on our work.

References

1. Buckley, C.: New york plans surveillance veil for downtown. The New York Times (July 2007), http://www.nytimes.com/2007/07/09/nyregion/09ring.html
2. McCahill, M., Norris, C.: CCTV in London. Technical report, University of Hull (June (2002)
3. Java Media Framework API: `http://java.sun.com/products/java-media/jmf`
4. Axis Communication: `http://www.axis.com/`
5. Alien Technology: `http://www.alientechnology.com/`
6. Deshpande, A., Nath, S.K., Gibbons, P.B., Seshan, S.: Cache-and-query for wide area sensor databases. In: Proceedings of the 2003 ACM SIGMOD, pp. 503–514. ACM Press, New York (2003)

7. Kulkarni, P., Ganesan, D., Shenoy, P., Lu, Q.: Senseye: a multi-tier camera sensor network. In: MULTIMEDIA 2005, Hilton, Singapore, ACM, New York, NY, USA (2005)
8. Gnawali, O., Jang, K.-Y., Paek, J., Vieira, M., Govindan, R., Greenstein, B., Joki, A., Estrin, D., Kohler, E.: The tenet architecture for tiered sensor networks. In: SenSys 2006, Boulder, Colorado, USA, pp. 153–166. ACM, New York, NY, USA (2006)
9. Ahmed, N., Kumar, R., French, R., Ramachandran, U.: RF2ID: A reliable middleware framework for RFID deployment. In: IEEE IPDPS, March 2007, IEEE Computer Society Press, Los Alamitos (2007)
10. Apostolopoulos, G., Kama, S., Williams, D., Guerin, R., Orda, A., Przygienda, T.: QoS routing mechanisms and OSPF extensions. IETF Draft, RFC 2676 (1996)
11. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In: MOBICOM, pp. 85–96 (2001)
12. Chatterjee, M., Das, S., Turgut, D.: An on-demand weighted clustering algorithm (WCA) for ad hoc networks. In: Proceedings of IEEE Globecom, IEEE Computer Society Press, Los Alamitos (2000)
13. PalChaudhuri, S., Kumar, R., Baraniuk, R.G., Johnson, D.B.: Design of adaptive overlays for multi-scale communication in sensor networks. In: Prasanna, V.K., Iyengar, S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, Springer, Heidelberg (2005)
14. Fallah, Y.P., Alnuweiri, H.: A controlled-access scheduling mechanism for QoS provisioning in IEEE 802.11e wireless LANs. In: Proceedings of the 1st ACM international workshop on Quality of service & security in wireless and mobile networks, pp. 122–129. ACM Press, New York (2005)
15. Alghamdi, M.I., Xie, T., Qin, X.: PARM: a power-aware message scheduling algorithm for real-time wireless networks. In: WMuNeP 2005. Proceedings of the 1st ACM workshop on Wireless multimedia networking and performance modelin, ACM Press, New York (2005)
16. Cho, S., Bettati, R.: Improving quality of service of tcp flows in strictly prioritized network. In: ACST 2006. Proceedings of the 2nd IASTED international conference on Advances in computer science and technology