VideoWeb: Design of a Wireless Camera Network for Real-time Monitoring of Activities

Hoang Nguyen, Bir Bhanu, Ankit Patel, and Ramiro Diaz
Center for Research in Intelligent Systems
University of California, Riverside
Riverside, California 92521 USA
Email: nthoang@cs.ucr.edu, bhanu@cris.ucr.edu, pankit@cs.ucr.edu, rdiaz005@student.ucr.edu

***Abstract*—Sensor networks have been a very active area of research in recent years. However, most of the sensors used in the development of these networks have been local and non-imaging sensors such as acoustics, seismic, vibration, temperature, humidity, etc. The development of emerging video sensor networks poses its own set of unique challenges, including high bandwidth and low latency requirements for real-time processing and control. This paper presents a systematic approach for the design, implementation, and evaluation of a large-scale, software-reconfigurable, wireless camera network, suitable for a variety of practical real-time applications. We take into consideration issues related to the hardware, software, control, architecture, network connectivity, performance evaluation, and data processing strategies for the network. We perform multi-objective optimization on settings such as video resolution and compression quality to provide insight into the performance trade-offs when configuring such a network.**

I. Introduction

We describe the development of a new laboratory called VideoWeb to facilitate research in processing and understanding video in a wireless environment. While research into large-scale sensor networks has been carried out for various applications, the idea of massive video sensor networks consisting of cameras connected over a wireless network is largely new and relatively unexplored.

The VideoWeb laboratory entails constructing a robust network architecture for a large number of components, including wireless routers and bridges, video processing servers, database servers, and the video cameras themselves. Hardware and equipment selection needs to take into account durability, performance, and cost. In addition, VideoWeb requires a number of software applications including those for data recording, video analysis, camera control, event recognition, anomaly detection, and an integrated user interface.

Challenges for the design of VideoWeb include creating a wireless network robust enough to simultaneously support dozens of high-bandwidth video cameras at their peak performance, providing power and connectivity to cameras, building a server farm capable of processing all the streaming data in real-time, implementing a low-latency control structure for camera and server control, and designing algorithms capable of real-time processing of video data.

The paper is organized as follows. In Section 2 we cover related work and contributions of this paper. Section 3 discusses the requirements and specifications used in designing the system. It discusses the technical challenges and solutions for actual implementation. Section 4 delves into characterizing the performance metrics from which to evaluate the system. Finally, Section 6 includes concluding comments and Section 7 suggests future and ongoing work.

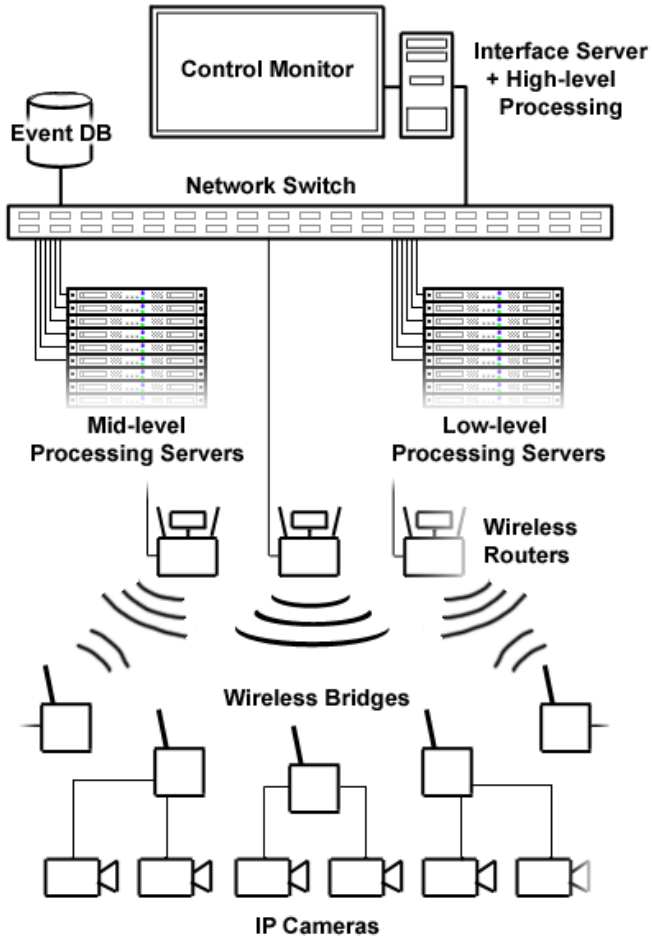


Fig. 1. Overall architecture. Top down: a single interface is used for direct control of any server/camera and high-level processing (e.g., user-defined face recognition). The server connects to a switch which hosts a database and joins two sets of servers: a series of mid-level (e.g., feature extraction) and low-level processors (e.g., detecting moving objects). The switch connects to routers which communicate with wireless bridges connected to the IP cameras.

978-1-4244-4620-9/09/$25.00

II. Related Work and Contributions

Many wireless camera platforms have been proposed [1], [2], [3] and emerging research in the design of wireless camera networks includes those with customized camera hardware nodes (e.g., CITRIC [4], eCAM [5]) including iMote2 and WiCa-based networks ([6], [7]), as well as networks with carefully-calibrated cameras ([8]).

This paper makes the following contributions:

1) We make the case for IP cameras and server-side processing by designing and implementing a system utilizing network cameras running on a software-reconfigurable server and network architecture. That is, while we use conventional IP cameras without on-camera processing, the configuration of the server-side processing is user-configurable and allows on-the-fly changes such as going from the default tiered-processing (e.g., low-level processing servers do object detection and send silhouettes to mid-level processors which generate object signatures and broadcast to high-level servers) to 1-to-1 camera-to-server processing which simulates the behavior of on-camera processing networks.
2) The VideoWeb system is designed to be flexible; every hardware component is interchangeable and replaceable. For instance, cameras can be heterogeneous (e.g., color and infrared videos) with different models from different vendors. The wireless routers and bridges can be interchanged or have their firmware updated to keep up with new wireless technologies, and the software architecture requires no specific server hardware. The software system has also been designed to be vendor-independent and cross-platform (e.g., any IP camera which can output Motion JPEG (M-JPEG) [9] is supported, a standard feature of many consumer and commercial cameras).
3) This paper describes performance metrics on which to evaluate a video network's performance and uses multi-objective optimization in order to discover Pareto-efficient settings for camera configurations.

III. Designing the Video Network

A. Architecture

The complete architecture (Figure 1) is comprised of a camera component, a wireless component, an application server component (e.g., database servers, digital video recording servers), and a processing component comprised of 3 levels: a set servers which process camera feeds at a low level (e.g., human detection, per-camera tracking), a set of servers which use this information for mid-level processing (e.g., feature extraction, multi-camera tracking), and a master server which uses this data for high-level processing and user control (e.g., task assignment, scene analysis, face recognition). The high-level server is also used as an interface for the network.

Since processing is not always oriented at a per-camera level (e.g., a processing task may be specific toward a group of cameras), a multi-tiered architecture is used to delegate camera control and processing responsibilities; the server architecture is designed as a 3-level tree hierarchy: a master high-level interface server communicates with a set of mid-level processing servers. Each of these process data received from a number of low-level processing servers. A server architecture physically connected in this fashion would entail cameras forwarding data to one set of servers which forward low-level data to another set of servers, and once more to the high-level server. To avoid the delay in data transfer, we do not physically connect server nodes directly to one another, but rather connect each node to a central network switch and implement this hierarchy in the networking configuration (i.e., DNS configuration) and communication-strategy in our software.

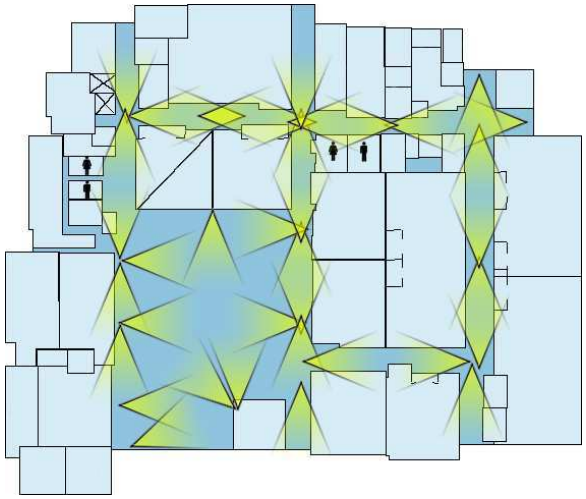


Fig. 2. Camera layout of the building.

The processing component handles all the control, interface, and processing servers. A key feature of the processing level is that all components can communicate with each other through a central network switch. An interface server is employed to allow users to view live or processed data from the cameras and to manually assign processing tasks (such as running a particular algorithm on some arbitrary number of cameras) from a central location. In the wireless component, cameras and servers are bridged through a single-hop wireless network using wireless routers connected to the servers to communicate with wireless bridges located throughout the building which connect to the cameras.

B. Cameras

1) Camera requirements: For this research, we use the VGA standard 640×480 pixels as the minimum video resolution requirement and a reasonable 20 frames per second as a threshold for acceptable real-time performance. In addition, we utilize digital IP cameras which provide a range of benefits such as streamlined processing (no digitizing required), relatively easy data storage, and simplified connectivity.

Cameras are to be installed in an indoor and outdoor building environment which includes locations such as remote open spaces exposed to rain and corridors void of sunlight. As

a camera network for long-term applications with year-round use, battery-powered cameras are not sufficient and we instead use network cameras with power adapters.

2) Camera selection: Among conventional pan/tilt/zoom (PTZ) cameras considered was the Panasonic WVNS202, Axis 214, and Axis 215 cameras. Besides cost, factors influencing camera choice include physical size, and availability of non-intimidating outdoor enclosures, and performance. The Panasonic cameras were deemed unsuitable after experiments which showed that the video stream begins to lag when the network becomes congested (Table I). That is, in the event of network throughput issues which limit cameras to low frame rates, instead of dropping frames, the Panasonic resends cached video frames stored in its buffer. The Axis cameras on the other hand, drops frames, maintaining a relevant video stream despite low frame rates. Between the two Axis cameras, the 215 was selected as the primary camera due to lower cost and lower mechanical latency when issuing PTZ commands.

3) Camera installation: Using a 45-degree field of view for the cameras, 37 locations were selected for complete coverage of the monitored building (Figure 2). As such, the network consists of 37 cameras: 36 Axis 215 PTZ cameras and a larger Axis 214 PTZ camera overlooking a courtyard. Each camera is capable of outputting a sustained 2.65 MB/second of Motion JPEG (M-JPEG) video at a peak of 30 frames per second when set to the maximum resolution of 704×480 pixels and a minimal compression setting of 0 (out of 100). This represents the maximum throughput and frame rate in an ideal environment (i.e., connecting to a camera via a direct ethernet connection and experiencing no frame drops). Other available resolutions of the cameras include 704×240, 352×240, and 176×120.

While the selected Axis 215 camera offers an outdoor dome enclosure, a dilemma was faced with the Axis 215 cameras, as there were no discreet outdoor enclosures for them; we had to find a way to make the cameras relatively weatherproof to withstand humidity and moisture. By choosing the Axis 215, we had to compensate for the lack of an available outdoor enclosure and improvise using the supplied flush-mount enclosures with smoked domes and surface-mount enclosures with clear domes, both designed for indoor installation. The solution was to use the surface-mount enclosures and make them waterproof by sealing the plastic seams with silicone sealant. In addition, the clear domes were interchanged with the smoked domes. The end result was a non-threatening camera dome suitable for surface-mounting at any of the 37 locations. Long-term effects of humidity, heat, and moisture on the cameras despite the sealed domes remains to be seen.

Electrical power was provided done by installing dedicated power supplies in two of the building's electrical rooms and running conduit to the camera cluster locations where power outlets were installed. Since we had full control of the power by using or our own power supplies, the cumbersome power adapters for the cameras were removed and the required power is supplied directly.

C. Wireless Bridges

1) Bridge requirements and selection: Since many IP cameras (the Axis 214 and 215 included) do not have built-in wireless connectivity, a wireless bridge is required to provide this functionality. As such, the wireless bridges serve a single purpose: connect the cameras to the routers. Since the camera locations are often situated in clusters, it is desirable if the bridges can support multiple clients (i.e., have more than 1 ethernet port). This quickly narrows down the selection. A conventional IEEE 802.11g bridge made by Buffalo was selected due to its support of 4 ethernet clients; IEEE 802.11n bridges were only available in 1-port versions at the time of selection. This paper does not delve into the pros and cons of individual wireless protocols, though literature on this specific topic has been recently made available [10]. Performance testing on the Buffalo bridges revealed no outstanding issues, but prolonged testing may show otherwise.

2) Bridge installation and configuration: The wireless bridges were installed throughout the building in the ceilings and localized in clusters where possible to better facilitate maintenance and troubleshooting concerns. In total, 19 wireless bridges are used to provide connectivity for the 37 cameras. Though the bridges have 4 inputs, we only use 2; we do not take full advantage of the bridges' connectivity capabilities for a reason. We originally planned to optimistically use 3 cameras per bridge, but found 2 cameras (streaming simultaneously with maximum video settings) was the limit each bridge could support without experiencing heavy frame loss. The bridges are configured to communicate with the routers using WPA-PSK encryption.

D. Wireless Routers

At a maximum of 2.65 MB/s per camera (or 5.3 MB/s from each bridge), the network may be generating over 98 MB/s of data at peak performance. Gigabit routers are used to handle the amount of expected traffic and IEEE 802.11n capabilities are chosen to facilitate future upgrades. We use Linksys WRT350N routers for the first iteration of the network. Routers are split into two clusters receiving from two indoor locations. In total, 5 routers handle traffic from 19 bridges. The routers are configured to assign local addresses to the cameras and port forwarding is used to address the cameras from the servers.

E. Servers

1) Server requirements: The most important component of the servers is the processor. We decided to go with a multi-core system in order to enable parallel data processing per computer. We have multiple cameras connecting to a single computer, so parallel processing would be very important. Also, with our server architecture we have 3 levels of processing. If later on this amount of processing power is insufficient, each computer should have a second vacant CPU socket for another processor to allow doubling the processing power of the server farm if necessary without increasing the physical footprint of the system. For uniformity and to facilitate maintenance, all processing servers have the same

TABLE I

Camera behavior can vary radically across vendors and models. Under congested network conditions for example, cameras may permanently drop frames or attempt to resend missed frames at the expense of live data. The Panasonic camera in this case output "smoother" video (fewer frame drops between two successive frames) under heavy network congestion (until its onboard cache is exhausted) at the cost of delays in upwards of 6 seconds.

	Panasonic WVNS202		Axis 215 PTZ	
Configuration	640×480 pixels, 0% compression		704×480 pixels, 0% compression	
Cameras per bridge	2	3	2	3
Frame delay (seconds)	< 1.0	$>$ **6.0**	< 0.5	< 1.0

hardware. The requirements of the applications which will run on the servers may be initially unknown, so it is desirable to have a reasonable amount of memory available (2GB) but also be expandable to higher memory density should the need arise.

An idea to use conventional desktop computers for data processing was quickly discarded due to the difficulty in physically scaling desktop computers or custom tower computers to a large number of cameras. Using even MicroATX cases would require a large amount of space to store the computers and would make moving the components/units particularly laborious and awkward. We instead opt for 1 height unit (1U) rack servers which can be housed in a single 42U rack enclosure with wheels for mobility.

2) Server construction and installation: In order to reduce contention over resources on the same machine from different camera processes, each processing server was specified with the following main hardware: an Intel Core 2 Quad Q6600 Kentsfield (2.4 GHz) CPU, 2GB DDR2 800 (PC 6400) memory, and a Western Digital SE16 250GB SATA hard disk. Though the Q6600 is not a true quad-core processor (2 dual-cores instead of 4 true cores), the support for additional threads is useful. Also, while we install 2GB for our initial setup, we also use motherboards which are expandable to 24GB of RAM. Gigabit ethernet cards are also selected to prevent any individual networking bottlenecks. Hard disks were given lower consideration, as most the processing nodes do mostly CPU processing and would not be storing data locally; the hard disks need only be sufficiently fast enough to run the operating system and RAM disks are setup in order to provide fast temporary storage for intermediate data. As such, conventional 80GB SATA hard drives are used. Application servers such as database or recording servers, on the other hand may emphasize larger and faster hard disks.

Thirty two identical servers were built and installed into a server rack. The building housing the servers fortunately has a suitable server room with adequate air conditioning and power connectivity. Kill A Watt devices were used to measure power consumption of the servers. The servers mentioned, for instance, peak at 198W/1.65A when starting up, use 132W/1.14A when idle, and consume 175W/1.54A under full load on all cores and hard drives. This data was then used to specify the uninterruptible power supplies (UPS) for the servers, which consist of four 2U APC Smart-UPS 2200VA/120V batteries. Testing showed the batteries capable of supporting 8 servers each at full load for 5 minutes and 45 seconds under an outage.

F. Software System

1) Software requirements: In order to implement the tiered processing scheme of the servers, the software needs to be both clients and servers to facilitate the sending and receiving of video traffic and camera controls. Mid-level servers, for instance, may need to broadcast a stream of processed data to the high-level server for viewing by the user, while at the same time being able to download cropped object images from low-level servers.

2) Software prototyping: The goal of the first software iteration was to control a networked camera using a customized program without the use of the supplied camera web interface or vendor-specific camera-management software included with most network cameras. One of the advantages of utilizing network cameras is that the camera control interface can be implemented through sending simple HTTP commands. To demonstrate this, a 10-line Python script was written for sending manual control commands to a camera. Once it was shown that it was easy to control the cameras, work started on a C++ application for the actual image processing.

3) Sample program - head tracking: The basic algorithm framework used for head tracking was based on a gradient and color-based tracker [11] with additional tweaks. The first implementation of this was done in C++ and MATLAB. Testing showed this program to be too slow for real-time processing, so a second iteration was written in pure C++ using OpenCV [12]. The tracker began with tracking synthetic object data consisting of randomly rotated rectangles of various sizes against a white backdrop. Once this stage was satisfactory, the next task was to grab live data from the camera.

Instead of relying on vendor-supplied software development kits (SDKs) which would have to be re-integrated into the processing software for potentially every type of camera, a generic camera controller was written. The SDK for the Axis cameras, for instance, relied on MFC-based [13] subroutines which would force development on Windows. In light of the amount of customization needed to incorporate a new SDK to do essentially the same things for different camera models, a generic cross-platform control framework was written from scratch. This control framework uses Boost.Asio [14] (a cross-platform socket wrapper) to directly send HTTP/1.1 [15] camera commands to a camera and uses the libavcodec library [16] to decode the streaming camera data. Using this approach, the software gains the benefit of being able to decode a large number of potential video streams and not just what a camera vendor has included with their SDK.

Networking communication between the three levels of

servers is also implemented with Boost.Asio. For instance, processed results performed by the mid-level servers is compressed and broadcast as an M-JPEG stream, which is then parsed and displayed by the interface server.

IV. Experiments for Performance Characterization and Optimization of the Video Network

A. Measurement software

Software that comes with most IP cameras ranges from small camera control programs to full surveillance station applications. However, even the most expensive or sophisticated of these vendor applications can be unsuitable since they are usually targeted toward security applications and recording, playback, and camera control are often their sole function. Evaluating performance using these applications is subjective and raises the need for our own statistic-recording implementation.

A custom program was written to fulfill this function. Given an IP address and port number, the application proceeds to:

1) Establish a connection with the camera
2) Attempt to download the M-JPEG video stream
3) Parse the stream into individual JPEG frames
4) Record real-time statistics about the stream

The program records a number of statistics and measurements including bandwidth, shortest lag between two frames, and the average, minimum, and maximum amount of bandwidth required for each frame. The implementation is in C++ and uses the generic control framework written earlier.

B. Optimizing Camera Configuration

Depending on the task or application, there are numerous "optimal" ways to configure a network. For instance, maximizing video resolution and quality may be paramount for biometrics, particularly in face recognition where a large number of pixels on the face is beneficial to identifying features. Surveillance and alarm systems, on the other hand, may find reliability more important. For instance, it may be more important that every moment is recorded with minimal skipping (not only for evidence in the event of an incident, but also because security applications often employ vision-based motion detection). Object tracking in turn, may benefit most by sacrificing resolution in exchange for a high sustained frame rate.

Configuring the network may consist of changing camera parameters (e.g., resolution, compression) as well as physical network parameters (e.g., number of cameras per bridge, number of bridges per router, number of routers per square foot). The later is helpful in introducing a metric for minimizing labor and monetary cost. We define 5 metrics for measuring camera network performance, the first two of which are used as configuration parameters.

1) *Resolution* (in pixels) - This measures the size of each video frame in pixels (the higher, the better). This parameter consists of 4 levels on the Axis cameras (704×480, 704×240, 352×240, and 176×120).
2) *Video compression* - This parameter represents the amount of *lossy* video compression applied to the video by the camera. For M-JPEG streams on the Axis cameras, this represents JPEG compression and ranges from 0 to 100 (the lower, the better). In our experiments, we test 5 of these levels (0, 20, 30, 60, and 100).
3) *Average frame rate* (in frames per second) - This measures the number of *complete* frames received per second, averaged over the duration of a measurement trial (the higher, the better). The frame rate may range from 0 to a maximum frame rate of 30 on the Axis cameras.
4) *Standard deviation of frame rate* - This measures the consistency of the video. For instance, there may be two video streams both 20 frames per second each, but the first may output a constant 20 frames per second while the second video may be sporadic and go from 30 to 0 to 10, back to 30 and so forth (but still average to 20 in the end). This metric is useful in evaluating the stability of the video (the lower the deviation, the better) and is measured by recording the delay between every two frames (in seconds with millisecond resolution) and calculating the standard deviation.
5) *Longest lag time between two complete frames* (in milliseconds) - This metric records the longest amount of time taken between any two consecutive frames (the lower, the better). This is insightful for evaluating a video stream's reliability (that is, it measures the longest amount of time a camera is "blind"). In addition to a depressed frame rate, this may be attributed to dropped/partial frames by the camera or data corruption/dropped packets undergone during transit.

C. Multi-objective Optimization Using Pareto Efficiency

We use the concept of Pareto efficiency to define which configuration of parameters is "better" than another. While this does not always tell a user which configuration should be used for a particular application, it serves to reduce the large number of possible configurations by showing which of those are usually "inferior"; a user only has to consider a configuration from the (potentially) much smaller Pareto set rather than every possible combination.

1) Inferiority and Non-Inferiority: Let M_1 be a vector of measurements of certain metrics for a camera and let M_2 be another trial of measurements on the same camera, but under a different parameter configuration. M_1 is said to be **inferior** to M_2 if and only if:

- every measurement in M_2 is equal to or outperforms the corresponding measurement in M_1
- one or more measurements in M_2 outperform the corresponding measurements in M_1

"Outperforms" is metric-specific and means "greater than" or "less than" depending on how the metric is defined (e.g., a *higher* frame rate outperforms a *lower* frame rate and a *lower* lag outperforms a *longer* lag). M_2 is said to be superior to

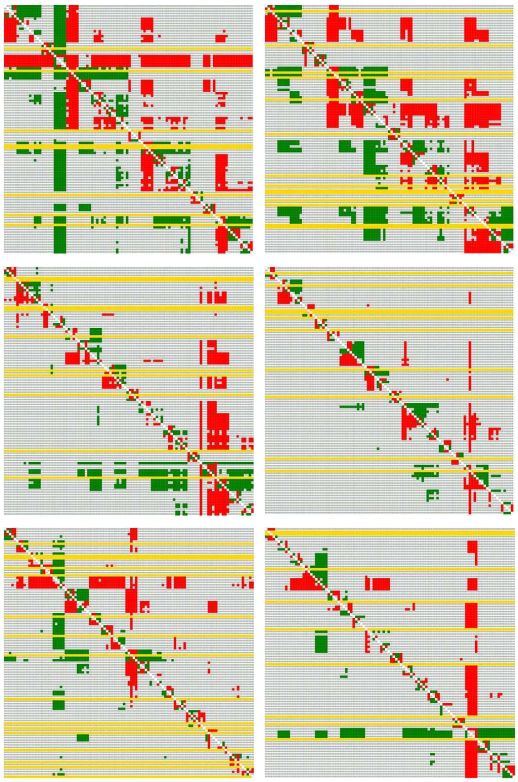


Fig. 3. Measurement comparison matrices for 6 cameras. While cameras may exhibit variable performance even when using the same configurations, some configurations may be inherently better than others and exhibit similar performance across the network. To discover these configurations, 100 trials are performed on each camera under a variety of parameter configurations (i.e., resolution and compression) and each recorded measurement is compared for Pareto efficiency against the other 99 trials. This results in a symmetric matrix where vertical and horizontal axes indicate the measurements M_i and M_j, respectively (i.e., the top-leftmost square in each matrix indicates the relationship of M_1 against M_{100}). Red indicates that a particular M_i is inferior to a particular M_j, green indicates superiority, and a solid horizontal yellow line denotes rows which are completely Pareto-efficient (i.e., either superior or non-inferior against all other 99 trials).

or *dominates* M_1 if M_1 is inferior to M_2. Finally, M_1 and M_2 are both said to be **non-inferior** if neither is superior nor inferior to one another.

In order for a measurement M_i to be **Pareto-efficient** (amongst a set), it must be non-inferior to every other measurement in that set. That is, it possesses at least one *advantage* over every other measurement when compared one-on-one (e.g., M_1 has higher frame rate against M_2, lower lag against M_3, ..., higher resolution than M_n). The Pareto set is the set of all Pareto-efficient measurements and ideally, allows a user to discard a large percentage of inferior parameter configurations from consideration when setting the cameras.

2) Data Collection: Data collection consists of varying the resolution and compression parameters and recording the measurements from 37 cameras. In total, we iterate through 4 resolutions (704×480, 704×240, 352×240, and 176×120) and 5 levels of compression (0, 20, 30, 60, and 100) each. Five measurement trials are captured for each of the 37 cameras per configuration (100 trials total per camera). Each trial consists of streaming from the camera for 600 frames or up to 2 minutes (whichever comes first).

Camera footage is tested at 5 various points in the day across all cameras. This exposes the data to a variety of video footage ranging from bright open areas with upwards of 20 moving people in the scene, to dark and grainy footage of cameras monitoring lonely halls.

After data collection is completed, each camera is optimized individually to minimize camera, bridge, or router bias. This is done in O(n^2) via exhaustive search (where n is the number of trials to compare), comparing each measurement to every other measurement on the same camera. With 20 configurations and 5 trials per configuration, each camera produces a symmetric 100×100 matrix. The resolution/compression pairs which result in the Pareto-efficient measurements for each camera are later aggregated against the entire network.

D. Evaluation Results

After over 100 hours of data collection at varying times of day across two weeks, the Pareto sets for all 37 cameras are calculated (see Figure 3 for sample matrices for 6 cameras). Considering only configurations in the Pareto sets eliminates (on average) approximately half of the tested configurations as inferior and redundant.

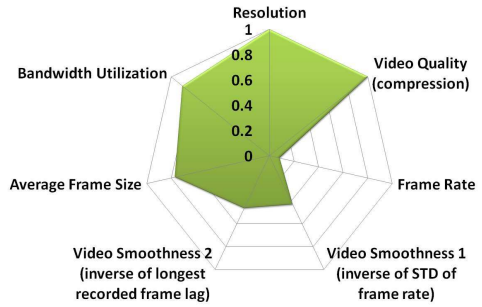
Compression Resoltion \	100	60	30	20	0
176x120	46%	**66%**	46%	**51%**	**74%**
352x240	34%	46%	26%	34%	**91%**
704x240	**51%**	29%	17%	**54%**	**97%**
704x480	34%	31%	**63%**	**94%**	**100%**

Fig. 4. Probability of configuration membership in a camera's Pareto set.

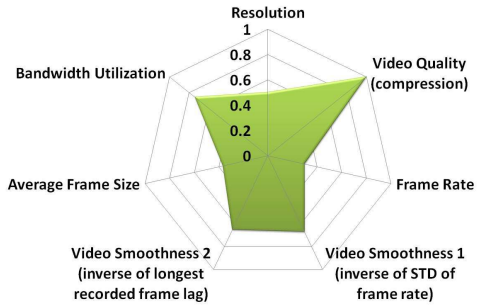
Further, after aggregating the resolution/compression pairs of the Pareto sets for the entire camera network, it is surprisingly found that *every* tested configuration is included in the Pareto set for at least one camera. This means there is no global consensus on any one configuration being inferior. Calculating the percentages, however, reveals that the cameras tend to prefer certain configurations over others (see Figure 4). This is in line with the previous observation that roughly half of the tested configurations are not preferred (less than a majority agreement between the cameras). It is not surprising to see higher percentages on configurations with either the maximum resolution or minimal compression since they already optimize at least one metric by definition. However, configurations such as 176×120/60% and 704×240/20% reveal local optimum which is potentially very useful for some practical applications of the video network. Using a more fine-tuned set of compression levels, we would likely be able to find more such points, aiding in the creation of a useful set of presets for specialized applications.

In order to evaluate the relative performance of the configurations, the measurements for each camera are normalized

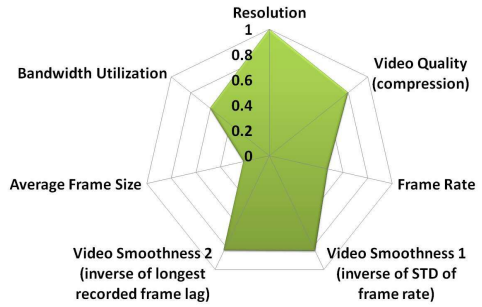
a. 100% of cameras: 704×480 pixels, 0% compression



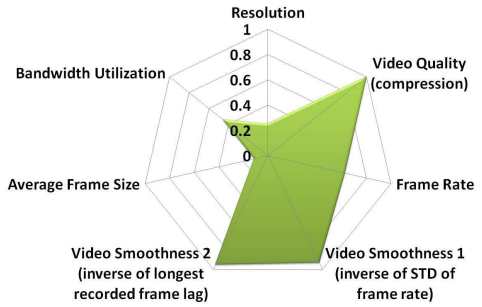
b. 97% of cameras: 704×240 pixels, 0% compression



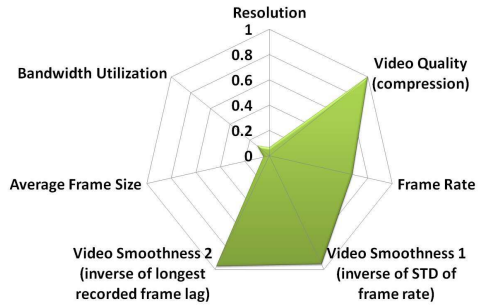
c. 94% of cameras: 704×480 pixels, 20% compression



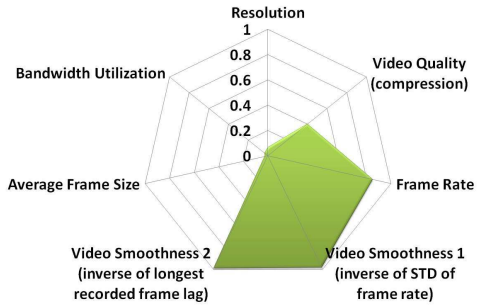
d. 91% of cameras: 352×240 pixels, 0% compression



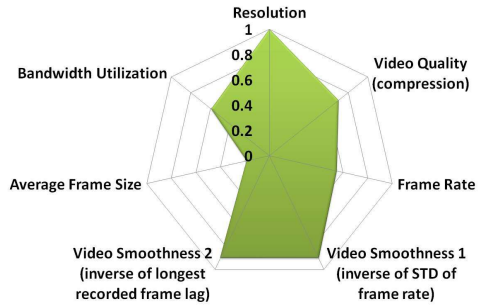
e. 74% of cameras: 176×120 pixels, 0% compression



f. 66% of cameras: 176×120 pixels, 60% compression



g. 63% of cameras: 704×480 pixels, 30% compression



h. 54% of cameras: 704×240 pixels, 20% compression

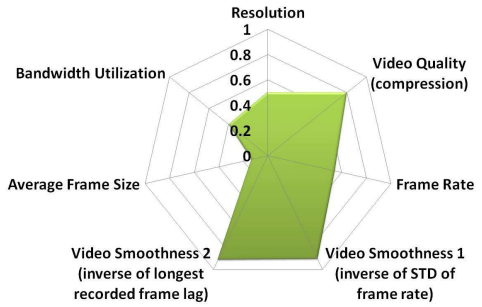


Fig. 5. The top 8 dominating camera configurations as chosen by 37 cameras. Graphs are ordered by the percentage of cameras in which the particular configuration was Pareto-efficient and all metrics are normalized to 1.0 across all cameras. Clockwise from the top: resolution ranges from 176×120 to 704×480 (higher is better), JPEG compression settings range from 0 to 100 (lower is better, so inverse is shown), and frame rates range from 0 to 30 FPS (higher is better). For measuring the "smoothness" of outputted video, the standard deviation of the frame rate (recorded at 1-second intervals) and maximum lag time between any two sequential frames is recorded (lower is better, so inverse is shown). Finally, average frame size (in bytes) and percentage bandwidth utilization is shown to demonstrate network throughput trade-offs.

across all measurements on the same camera and then averaged on a per-configuration basis across all cameras using the same configuration. Figure 5 shows the relative performance of the top 8 configurations for the entire network.

Intuitively, increasing either the resolution or decreasing the compression (resulting in higher bandwidth) has the effect of a reducing the frame rate, producing a more discontinuous video stream, and increasing the maximum lag time.

V. Conclusions

We have designed an software-reconfigurable architecture for a wireless network of a large number of video cameras and implemented a working system by building the servers, installing the cameras, writing the software, and configuring the network to support it. Further, we gained insight into configuring the network's cameras by defining a set of metrics and discovering Pareto-efficient camera configurations by performing multi-objective optimization on a large volume of real data recorded by the system.

The idea persists that if one has a camera network with 30 FPS cameras, one will be able to obtain the said 30 frames per second regardless of network configuration or parameters. Though this may be true in a controlled test environment, the performance expectation should not be so optimistic for real-world implementations. Even using the most preferred Pareto-efficient configurations on a non-congested network, it is shown that frame rates will most certainly suffer and that tradeoffs must be made.

In line with this confirmation is the need to emphasize that *partial frames are important*. Rather than having algorithms which assume that the data consists entirely of complete video frames (and are only capable of processing such frames), real-time computer vision algorithms should take advantage of as much information as is available to them; a stream of partial frames which may only be missing the last few rows of data can still be tremendously useful for a number of applications.

VI. Future Development

As is, the testbed system effectively assumes a network with 1 camera per bridge (since no two cameras from the same bridge are simultaneously tested). As mentioned earlier, it would be advantageous to be able to add cost metrics by altering the number of cameras per bridge and number of bridges per router. These parameters would play an important role by giving insight into the capabilities of the networking equipment itself which lends well in the event of network expansion (e.g., one would be able to determine how much the frame rate would suffer on a given resolution and compression if an additional camera was added to every bridge).

Speed improvements will be made by making the application multi-threaded. Currently, all image processing is done linearly in a single thread; implementing a multi-threaded algorithm would take advantage of the multi-core servers running the application, resulting in significantly lower latency in processing a single image.

In addition, a touch-screen interface application will be written to control the system. The low to high-level processing applications can be entirely command-line-based, but a web-based user interface will be used to interact with various levels of applications. Such an interface will be able to take advantage of a computer touch screen to select cameras, send targeted commands to specific cameras, and if desired, automatically control a set of cameras.

VII. Acknowledgments

This work was supported in part by NSF grants 0622176, 0551741 and ONR grants N00014-07-1-0931 and Aware Building.

References

[1] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Comput. Netw.*, vol. 51, no. 4, pp. 921–960, 2007.

[2] W.-T. Chen, P.-Y. Chen, W.-S. Lee, and C.-F. Huang, "Design and implementation of a real time video surveillance system with wireless sensor networks," in *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, May 2008, pp. 218–222.

[3] H. Park, J. Burke, and M. B. Srivastava, "Design and implementation of a wireless sensor network for intelligent light control," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 370–379.

[4] P. Chen, P. Ahammad, C. Boyer, S.-I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, A. Yang, C. Yeo, L.-C. Chang, J. Tygar, and S. Sastry, "Citric: A low-bandwidth wireless camera network platform," in *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, Sept. 2008, pp. 1–10.

[5] C. Park and P. H. Chou, "eCAM: ultra compact, high data-rate wireless sensor node with a miniature camera," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2006, pp. 359–360.

[6] T. Teixeira, D. Lymberopoulos, E. Culurciello, Y. Aloimonos, and A. Savvides, "A lightweight camera sensor network operating on symbolic information," in *First Workshop on Distributed Smart Cameras 2006, held in conjunction with ACM SenSys 2006*, November 2006.

[7] R. Kleihorst, A. Abbo, B. Schueler, and A. Danilin, "Camera mote with a high-performance parallel processor for real-time frame-based video processing," in *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, Sept. 2007, pp. 69–74.

[8] M. Quinn, R. Mudumbai, T. Kuo, Z. Ni, C. D. Leo, and B. S. Manjunath, "Visnet: A distributed vision testbed," in *ACM/IEEE International Conference on Distributed Smart Cameras, 2008*, Sep 2008, pp. 364–371. [Online]. Available: http://vision.ece.ucsb.edu/publications/quinn_icdsc_2008.pdf

[9] Network Working Group, "RFC 2435: RTP payload format for JPEG-compressed video," 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2435.txt

[10] N. Li, B. Yan, and G. Chen, "Measurement study on wireless camera networks," *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, pp. 1–10, Sept. 2008.

[11] S. Birchfield, "Elliptical head tracking using intensity gradients and color histograms," *CVPR 1998*, pp. 232–237, 1998.

[12] G. Bradski, "Open Computer Vision Library (OpenCV)," 1999-2009. [Online]. Available: http://opencv.willowgarage.com/

[13] I. Microsoft, "Microsoft foundation classes MFC," 1992-2008. [Online]. Available: http://msdn2.microsoft.com/en-us/library/d06h2x6e(VS.80).aspx

[14] C. M. Kohlhoff, "Boost.Asio: a cross-platform c++ library for network and low-level I/O programming," 2008. [Online]. Available: http://www.boost.org/doc/libs/1_37_0/doc/html/boost_asio.html

[15] Network Working Group, "RFC 2616: Hypertext transfer protocol – HTTP/1.1," 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[16] FFmpeg Team, "libavcodec: audio/video codec library," 2009. [Online]. Available: http://ffmpeg.mplayerhq.hu/