TOPOLOGY ESTIMATION FOR THOUSAND-CAMERA SURVEILLANCE NETWORKS

Henry Detmold, Anton van den Hengel, Anthony Dick, Alex Cichowski, Rhys Hill,
Ekim Kocadag, Katrina Falkner and David S. Munro
{henry,anton,ard,alex,rhys,ekim,katrina,dave}@cs.adelaide.edu.au

The Australian Centre for Visual Technologies
School of Computer Science
The University of Adelaide

ABSTRACT

Surveillance camera technologies have reached the point whereby networks of a thousand cameras are not uncommon. Systems for collecting and storing the video generated by such networks have been deployed operationally, and sophisticated methods have been developed for interrogating individual video streams. The principal contribution of this paper is a scalable method for processing video streams collectively, rather than on a per camera basis, which enables a coordinated approach to large-scale video surveillance. To realise our ambition of thousand camera automated surveillance networks, we use distributed processing on a dedicated cluster. Our focus is on determining *activity topology* – the paths objects may take between cameras' fields of view. An accurate estimate of activity topology is critical to many surveillance functions, including tracking targets through the network, and may also provide a means for partitioning of distributed surveillance processing. We present several implementations using the *exclusion* algorithm to determine activity topology. Measurements reported for the key system component demonstrate scalability to networks with a thousand cameras. Whole-system measurements are reported for actual operation on over a hundred camera streams (this limit is based on the number of cameras and computers presently available to us, not scalability). Finally, we explore how to scale our approach to support multi-thousand camera networks.

Index Terms— Large-scale surveillance networks, Software architectures, Collaborative position discovery

1. INTRODUCTION

Video surveillance networks serve a number of purposes including the protection of major facilities from terrorism and other threats. At the hardware level, it is now possible to build thousand camera networks at reasonable cost, using standard IP networking devices and IP video cameras. However, monitoring surveillance networks through human inspection is both expensive and remarkably ineffective: trained operators lose concentration and miss a high percentage of significant events after only a few minutes. Consequently, there is a need for *software* for *automated* video surveillance [1], to assist human inspection in the operation of surveillance networks.

Most research in this area concentrates on computer vision algorithms required to detect and interpret activity in video. This previous work is limited to networks of at most tens of cameras. In this paper our focus is on networks with between one hundred and one thousand cameras. Specifically we have constructed and measured on-line approaches to the key *activity topology* problem for such networks. We measure scalability up to one thousand cameras, report whole-system results for over one hundred cameras and explore how our approach can be adapted to scale beyond one thousand cameras.

2. ACTIVITY TOPOLOGY

The *activity topology* is a graph describing the observed (*i.e.* past) behaviour of target objects in the network. The edges in the activity topology describe the paths surveillance targets take between cameras' fields of view (nodes in the topology). The edges may be weighted: with probabilities describing the likelihood of targets moving from one camera to the other, the density of such movement and/or the mean time taken to so move. The current estimate of activity topology can be used to predict future behaviour of targets from their current positions. This predictive function is useful in a number of higher-level functions, including:

- *Inter-camera tracking* – Statistical approaches for tracking a target within a camera's field of view fail when the target leaves that field of view. In such cases, a search is needed to discover in which camera the target next appears. In the absence of activity topology, all other cameras' fields of view must be searched (which is $O(n^2)$ for n cameras). Activity topology restricts the cameras to be searched to those adjacent to the current camera, and may also enable the search to be prioritised according to likelihood of the target's next appearance.

This work was supported by ARC Discovery Grant DP0770482 and the Government of South Australia PSRF scheme.

- *Target following virtual cameras* – Humans monitoring a surveillance network find it extremely disorientating to follow a target moving between fixed cameras. Nevertheless, fixed cameras are a pre-requisite for most video processing, which is greatly degraded when cameras are moving. A solution is to provide operators with "virtual cameras" that follow each target, switching between physical cameras automatically. The activity topology is use to determine candidates for the next camera to switch to.

- *Camera placement optimisation* – High density of movement along an activity topology edge militates for the placement of additional cameras between the cameras connected by the edge. In contrast, cameras having no incident edges or only edges with low movement density should probably be moved to locations in which they can be more effective.

In addition to its importance in supporting *functional* requirements such as the above, activity topology potentially provides a basis for partitioning the processing in a surveillance network, thus providing a generic tool for achieving the *systemic* requirement of *scalability* in distributed surveillance processing. The idea is that the cameras in the network are partitioned into *near strongly connected components* within the activity topology. These are sub-graphs of the topology where there are many edges between the nodes within the sub-graph, and few edges between sub-graphs. This partitioning enables a *divide and conquer* approach whereby most processing (typically with $O(n^2)$ time and space complexity for n cameras) occurs within sub-graphs, with results from sub-graphs then merged, via algorithms having complexity determined by the (small) number of edges between sub-graphs.

There are a number of reasons why activity topology demands an online solution, as opposed to determining topology off-line prior to surveillance network commissioning:

1. The system must remain operational whilst the activity topology changes, as cameras are added, removed and repositioned, and other changes occur in the facility (*e.g.* a door is unlocked). Such changes are more common than might be expected, Also, apparently minor changes can cause major alterations in the topology.

2. By definition, an off-line process must run to completion before the topology may be used. It may take a long time for such a process to obtain information about areas that are seldom visited, simply due to paucity of information about such areas. In contrast, an on-line process will rapidly acquire information about frequently visited areas, and make that information immediately available, whilst continuing to acquire information about seldom visited areas, as and when those areas are visited.

Another possible alternative is to rely on human operators to input the activity topology. Practical experience indicates this is very unreliable; in addition to the problems above (relevant since human input is inherently an off-line approach), humans' ability to predict the activity topology from the camera configuration is quite poor, in part because recording of the actual spatial relationships between cameras (and other feature) is rarely sufficiently accurate, and also because such relationships only partly determine activity topology, since they do not account for autonomy in the behaviour of people and other objects under surveillance.

3. SURVEILLANCE NETWORK PERFORMANCE

Our interest is in surveillance networks for *threat detection*. Threat detection surveillance networks operate continuously in an on-line mode where they attempt to detect undesirable behaviour from observations in video streams, and to bring this to human operators' attention as soon as possible. Software for such surveillance networks has at least two overall performance goals:

- *Efficiency* – to maximise the *surveillance capacity* (number of cameras that the network can support) on given *processing capability* (CPUs, memory, network *etc.*).

- *Scalability* – to permit increased surveillance capacity through the addition of processing capability, with the increased surveillance capacity ideally in proportion to the increase in processing capability.

Scalability typically reaches a limit, presenting a third goal:

- *Scalability Limit* – to maximise the surveillance capacity at which it ceases to increase surveillance capacity simply by procuring increased processing capability.

To evaluate the performance of automated surveillance network software in relation to these goals, we define:

1. *Steady-state Throughput* – the quantity (*e.g.* frames per second) that a given surveillance network can process in on-line mode, once it has reached steady state operation. This is the key metric; essentially it determines a surveillance network's surveillance capacity.

2. *Detection Delay* – the delay between physical occurrence of an event captured in a video stream and the incorporation of that event within the surveillance computation (*i.e.* the activity topology estimate is updated to reflect the event). Whilst surveillance networks are not real time systems (at least when used for threat detection), detecting intruders half an hour after they have left the building is not acceptable.

Our experiments involve measuring the steady state throughput of different network configurations in terms of standardised video streams with 320 by 240 pixel frames and subject

to meeting constraints in terms of *frame rate*, *detection delay* and *memory usage*. Specifically:

1. Frame rate of at least 10 frames per second per camera.
2. Detection delay of at most 10 seconds.
3. Each processing node must have a known maximum memory requirement, which is not exceeded no matter how long the system runs.

The memory requirement constraint is needed so that we can provision hardware with enough memory to avoid paging (and the consequent dramatic deterioration in performance).

4. EXCLUSION

The basis of exclusion is the very simple observation that:

> *If one camera's field of view is occupied and another camera's field of view is simultaneously unoccupied, then the two cameras cannot be observing the same space.*

Occurrences of this situation are termed *exclusions* and constitute negative evidence refuting potential connections in activity topology. This simple *exclusion principle* can be developed into a practical activity topology estimation technique, even for the case where there is no overlap between the fields of view of adjacent cameras.

4.1. Segmentation and Spatial Padding

Instead of considering each camera as a unit of view, we segment each field of view into a grid of *windows*, and then apply exclusion between windows, rather than between cameras. In the extreme, windows would be individual pixels, but we have found that 40 x 40 pixel windows provide enough resolution to accurately recover overlapping camera regions. At each point in time, we perform background subtraction and lowest visible extent calculations to determine the *occupancy* status of each window. We do not currently include other measurements such as appearance or optical flow, as these can be unreliable for small targets and the challenging environmental conditions in which surveillance cameras often operate.

Since windows are not points, it is possible that two windows actually observing the same scene can be only partially overlapped such that one is occupied and the other simultaneously unoccupied, leading to a false exclusion of the actual correspondence between the windows. To deal with this, we calculate (spatially) padded occupancy for each window, whereby a window is considered occupied if it or any of its adjacent windows (within the same camera) are occupied. Then we define an exclusion to arise between windows w_1 and w_2 only if w_1 is true in (non-padded) occupancy data and w_2 is simultaneously false in padded occupancy data. Notice that this operation, which we write $w_1 \ominus w_2$, is asymmetric.

4.2. Temporal Padding

The technique previously described will detect adjacency of cameras having overlapping fields of view, and thus estimate activity topology for networks of such cameras. Whilst this is useful, in many surveillance networks there are few if any overlaps between cameras, and in fact regions between cameras which are not within any camera's field of view.

To deal with this scenario, we apply temporal padding to the (spatially) padded occupancy data, such that in the temporally and spatially padded data for a point in time, a window is considered occupied if is occupied in the spatially padded data at any point in time within some tolerance parameter. We then redefine the right had side of the exclusion test ($w_1 \ominus w_2$) to use the temporally and spatially padded data. Notice this also overcomes clock skew between cameras.

4.3. Accumulation of Evidence for Exclusion

Whilst a single exclusion is in principle enough to rule out a potential relationship between windows, in practice the calculation of occupancy is not perfect and it is wise to be more conservative. Therefore, we consider each exclusion ($w_1 \ominus w_2 = True$) as evidence against adjacency of the windows w_1 and w_2. To accumulate this evidence we count both exclusion opportunities (those times when w_1 is occupied) and detected exclusions (those points in time when $w_1 \ominus w_2 = True$) and consider the ratio between the two (note that the count of detected exclusions must always be less than or equal to the count of exclusion opportunities). Ratios close to 1 for sufficiently large exclusion opportunities provide strong evidence against adjacency of the given windows.

5. INITIAL OFF-LINE SYSTEM

The initial system architecture is shown in Figure 1. Excepting the cameras, all processing occurs on a single computer.

5.1. Implementation

As seen in Figure 1, processing conceptually occurs in a pipeline. However, since the intention at this stage is to verify functionality and establish the ratio of processing requirements for the different stages, we actually run the first three stages on video footage, store the results in files, then run the last two stages on those files. This is not an on-line system in any sense so we do not measure its throughput.

After decompressing JPEG encoded frames sent by cameras, the system uses the Stauffer and Grimson *background subtraction* method [2] to identify the foreground component of each frame. The next pipe-line stage, *occupancy detection*, derives a single occupied window for each foreground blob, using a connected components approach and taking the midpoint of the lower edge of the bounding box for the blob.

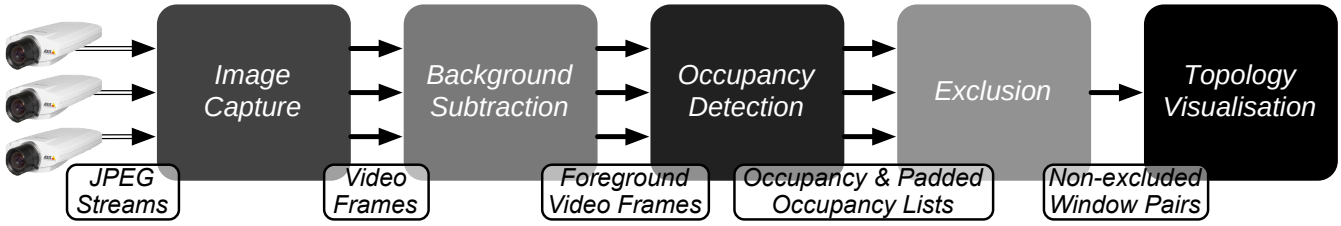

Fig. 1. Baseline System Architecture.

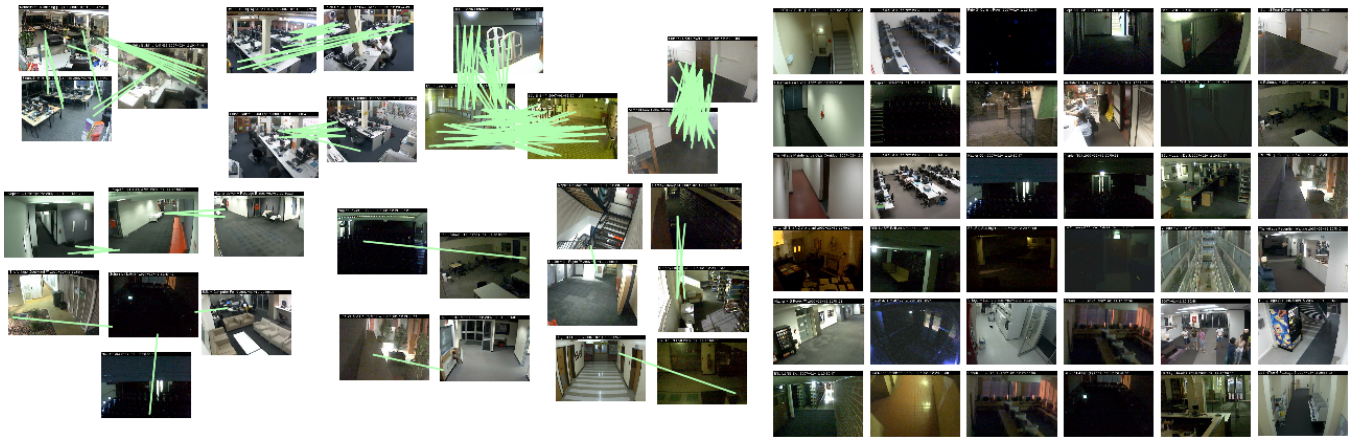

Fig. 2. Video feeds after running exclusion on one hour of footage from cameras spread across a university campus. The cameras are arranged on screen so that related cameras are near each other.

This midpoint corresponds approximately to the lowest visible extent of the blob. This pipe-line stage generates both *occupancy* and *padded occupancy* data for the next stage.

The next pipe-line stage, *exclusion* is the heart of the approach. Two integer matrices maintain exclusion counts, E_{ij} and exclusion opportunity counts O_{ij} for each pair of windows. All matrix elements are initialised to 0. O_{ij} is incremented whenever it is *possible to evaluate* the exclusion $w_i \ominus w_j$, which requires that w_i is occupied and that occupancy data are available (*e.g.* not missing due to a camera being off-line *etc.*) for all points needed to calculate padded occupancy for w_j. E_{ij} is incremented if O_{ij} is incremented and $w_i \ominus w_j = True$. Thus we can calculate the likelihood of non-exclusion between w_i and w_j as:

$$C_{ij} = \frac{O_{ij} - E_{ij}}{O_{ij}} \tag{1}$$

However, when windows are occupied only a small number of times, the above is dominated by noise, so we use:

$$C'_{ij} = C_{ij} \times \min\left(1, \frac{\log(O_{ij})}{\log(O_{ref})}\right) \tag{2}$$

where O_{ref} is a number of detections empirically determined to result in reliable exclusion calculation. We set this to 20 in our experiments.

Notice that the memory requirements for both E_{ij} and O_{ij} are $O(n^2)$ integer counts for n cameras. Two strategies are implemented to reduce actual memory requirements:

1. Byte-sized counts are used instead of words. When a O_{ij} count overflows, it is halved, as are any other counts in the same row which are greater than half the maximum value. To maintain the $E_{ij} <= O_{ij}$ invariant, the corresponding O_{ij} counts are also halved.

2. Run-length encoding is used to provide sparse representations of the E_{ij} and O_{ij} matrices.

The final pipeline stage, *topology visualisation* infers adjacency between w_i and w_j when $C'_{ij} > C^*$ and $C'_{ji} > C^*$, with C^* set to the empirically determined threshold 0.8. These inferred adjacencies are then used to visualise edges in the topology, such as shown in Figure 2, which we have manually verified accords with the actual overlap between cameras. The footage shown in Figure 2 contains indoor and outdoor areas, and periods of both high activity (when students are moving between lectures) and low activity (lectures in progress).

5.2. Performance Characteristics

We applied this initial system to process 2 hours of footage from each of 83 cameras at approximately 15 frames per sec-

ond, then measured the fraction of time taken by the different pipe-line stages. Processing for the first three stages took 51 hours, whereas the last two stages took 11 minutes.

6. DISTRIBUTED DETECTION PIPELINES

As described previously, the off-line system spends the majority of its time performing background subtraction and other processing on individual video streams. This leads to the obvious strategy of replicating the detection pipeline that runs this processing to multiple processors, so that streams from different cameras are processed in parallel on different computers. In addition, this system operates in on-line mode.

This architecture could also accommodate smart cameras running the detection processing stages. We have not yet investigated this option however, as current cameras do not have the processing capacity required.

6.1. Implementation

The distributed detection pipe-lines (DDP) system, shown in Figure 3 is a relatively straightforward adaptation of the baseline system. Essentially the first three stages of the pipeline are replicated to multiple detection nodes and the final two pipe-line stages are implemented on a central node, with TCP/IP connections carrying the data from the detection nodes to the central node. The complications are:

1. A file format is required for transmission of the occupancy data over the network. We use an XML format to provide flexibility for future extension of our system.

2. Occupancy data for a given time point arrives from different detection nodes at different times. Therefore we buffer incoming data in T_r second time ranges, adding incoming data to the appropriate range, and pass the whole of a range on to the exclusion pipeline stage every T_r seconds (using the central node's wall clock).

3. Once a given time range is processed, we ignore any further occupancy data within that range (so the later arrival of that data has no effect on either O_{ij} or E_{ij}). To ensure that there is a high probability that occupancy data arrives in time to be counted, the central node does not start to process the first range until T_d seconds after it starts.

A consequence of the above is that, (with empirically determined values $T_r = 2$ and $T_d = 5$), we have a bounded detection delay of $T_r + T_d = 7$ seconds, at the cost of ignoring data that are too late to be processed within its proper range. We measure and report the fraction of this ignored data.

We use a dedicated cluster to implement the distributed detection pipe-lines system, consisting of 16 2.0Ghz dual-core Opteron processors as the detection nodes, a single 8 core 1.86 Ghz Xeon as the central node and a Gigabit Ethernet switch connecting them together.

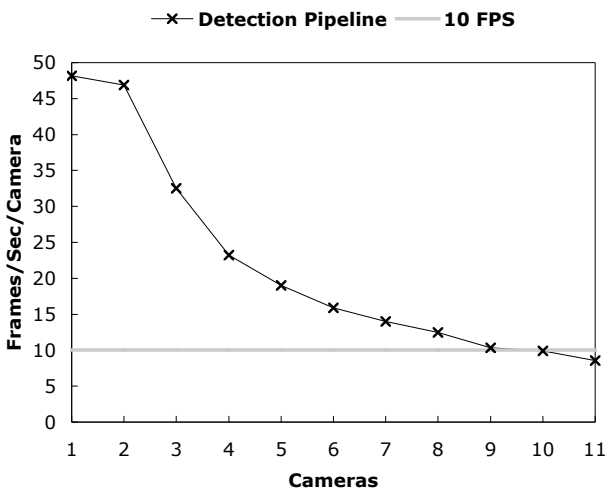

Fig. 4. Detection Pipe-line Capacity

6.2. Detection Pipe-line Performance

Detection pipe-line nodes operate independently of the central node, so the first task in establishing the system's performance is to measure the throughput of a detection node (on 2.0Ghz dual-core Opteron processors). Figure 4 shows the total frame rate achieved when processing a number of hour long, 10 FPS camera streams. This test involves running an instance (process) of the detection pipe-line per stream, all on one detection node . It can be seen that each pipe-line loses the ability to meet the 10 frames per second per camera constraint beyond 9 cameras – that is the system's *surveillance capacity* is at most 9 cameras.

6.3. System Performance – Measurement Complications

So as to conduct repeatable experiments and assist us in verifying that the network is operating correctly in all instances, our performance measurements use previously recorded video footage instead of live camera data. The extent to which this renders the measurements artificial is quite limited: the main problem is that it becomes possible to obtain input faster than real-time (particularly when footage is missing within a stream). To prevent this, when taking whole system measurements, we introduce artificial delays in detection pipe-lines to ensure that frames are not input faster than they would be in real-time. The overall effect of these delays is to reduce measured surveillance capacity below the actual capacity.

We currently have access to about 80 cameras. Therefore, we replicate footage where more than that number of inputs are required. Again, the effect is to reduce measured surveil-

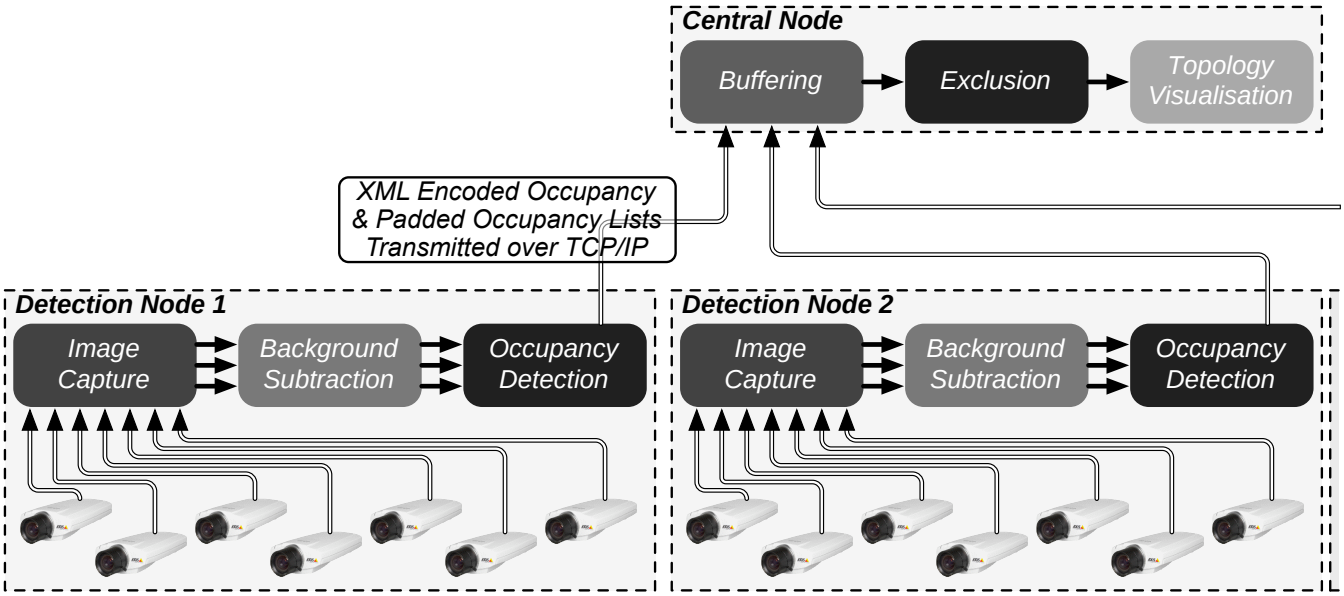

Fig. 3. Distributed Detection Pipe-lines (DDP) Architecture.

lance capacity below the actual capacity, because the exclusion algorithm is required to do extra work in accounting for the artificially exact correlation between copies of a stream.

6.4. System Performance – Results

Configuring each detection node with 7 cameras, we are able to instantiate a system with 15 detection nodes and thus 105 cameras. Measurements verify the following constraints, implied by the implementation, are actually met:

- Detection delay less than 10 seconds.
- Detection node memory usage is bounded (and trivial).
- Central node memory usage is bounded by the 400MB total size of the O_{ij} and E_{ij} matrices.
- Output frame rate (from exclusion on the central node) is 10 FPS, and input frame rate is 10 FPS/camera, hence throughput is 10 FPS/camera.

As noted previously, the detection delay constraint is automatically met in this system, but at the cost of late data being ignored and thus and potential loss of accuracy. The fraction of late data for each minute during an 57 minutes' processing of the 105 camera system is shown in Figure 5. This remains below 1.2% for the whole period, thus is unlikely to have a significant adverse effect. The spike at 38 minutes occurs on each run of the experiment, and reflects events in the camera network when the footage was collected (namely, students moving between lectures between 5 and 8 minutes past the hour). The figure also shows actual memory usage during this period; at considerably below 400MB, this demonstrates the

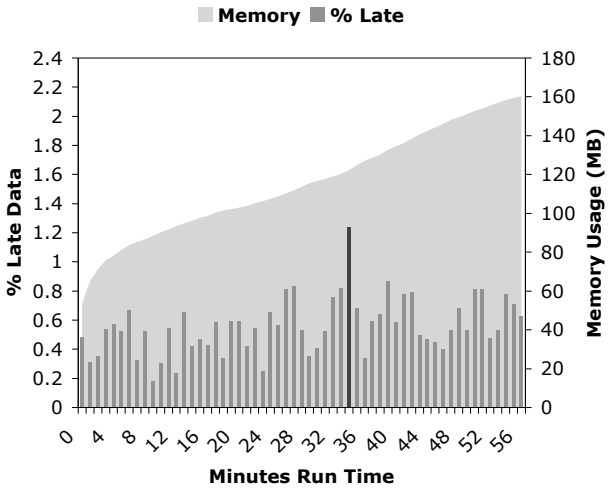

Fig. 5. 105 Camera DDP System – Constraints Maintained

effectiveness of the sparse matrix representation using run-length-encoding. These results establish that our system has measured surveillance capacity of at least 105 cameras.

7. MULTI-THREADED EXCLUSION

The distributed detection pipelines system achieves scalability in that its capacity can be increased simply by adding detection nodes to process data from additional cameras. The limit on scalability is the central node. In order to exploit the 8 processor cores on the central node, we developed a multi-

threaded implementation of the central node processing.

7.1. Implementation

Within exclusion processing for a time range, eight worker threads (one per core) are created, and these then process separate regions in space. This yields a speed-up of 5x for the inner loop and 3x for overall operation of the central node.

7.2. Performance

Figure 6 shows throughput results for the central node, in both single-threaded and multi-threaded versions. These measurements involve the topology estimator operating on files containing saved occupancy data. The points at which the curves cross the 10 FPS threshold suggest approximately a 400 camera scalability limit for the single-threaded implementation and just under 1,000 camera scalability limit for the multi-threaded implementation.

Interestingly, the scalability limit (for the multi-threaded implementation) is determined as much by memory usage as by CPU. With 1000 cameras (each having 108 windows), there are $(108 * 1000)^2 = 11,664,000,000$ byte-size counts required in each of the O_{ij} and E_{ij} matrices. This gives a maximum memory requirement of 24GB, which is close to the practical limit of affordable memory with current technology (our current central server has 12GB), and fixes the scalability limit for this approach at about 1000 cameras.

8. TOWARDS DISTRIBUTED EXCLUSION

Multi-threading the exclusion based topology estimator raises the limit to which the network can scale, but in order to pursue further improvements we need to distribute the exclusion algorithm. Our fundamental (but as yet, unverified) hypothesis is that strongly connected components within the activity topology provide natural partitions for processing. It follows that an activity topology estimate provides a basis for a divide-and-conquer approach to scaling surveillance computations. Clearly this requires elaboration when the computation to be partitioned is the activity estimator. We observe:

1. The task of estimating *de novo* the activity topology of a very large network (*e.g.* million camera) is unlikely in practice. Instead, a very large network is likely to arise by repeated extension of a previous smaller network, and so activity topology can be estimated incrementally, at each step using the previous estimate of activity topology to partition processing for the large number of previously existing cameras, and unpartitioned processing for the small number of additional cameras.

2. Whilst the exclusion algorithm requires comparison of $O(n^2)$ window pairs (for n windows) at each time point, the only effect of ignoring a large subset of the pairs at any time point is a failure to detect exclusions indicated by that data. Over time, assuming fairness in selecting the subset to be inspected, this equates to detecting exclusions more slowly than would be the case if all pairs were inspected.

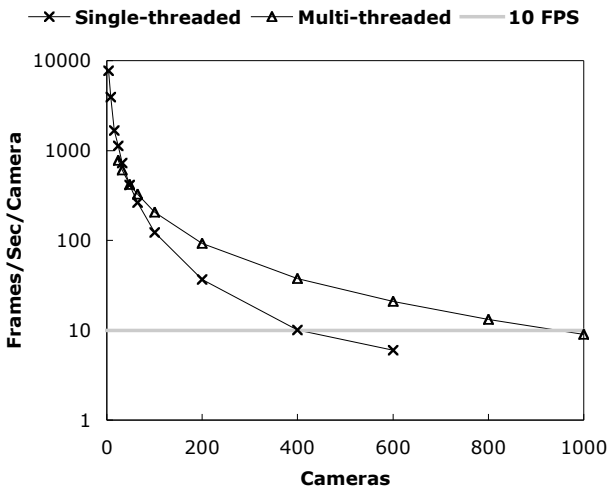

Fig. 6. Central Node Capacity

These observations lead to the following approach:

1. A bootstrap partitioning is generated, for example from the layer 2 switching topology of the IP network to which the cameras are attached, or, if the initial number of cameras is small enough, by running an unpartitioned exclusion algorithm on that network, then partitioning according to the activity topology estimate.

2. Once bootstrapping is complete, and thereafter, the existing cameras in the network are partitioned, albeit not necessarily optimally. In addition to running the normal exclusion algorithm within each partition, the union of the occupancy data for all camera windows in the partition is calculated for each point in time.

3. As new cameras are brought on-line, they are added to a *nursery*, in which they remain for some time (perhaps several days). The normal exclusion algorithm is run on all the cameras in the nursery. In addition, a trivially modified exclusion algorithm is run between each nursery camera window and the union occupancy for each existing partition, an approach we term *union exclusion*.

4. Periodically, the nursery is cleared by moving the nursery cameras out to other partitions. Where the activity topology within the nursery identifies one or more subsets of strongly connected cameras, new partitions are

formed containing these subsets. Where a nursery camera is not part of such a subset, it instead joins the existing partition from which it is least strongly excluded (based on the union exclusion calculation).

5. Occasionally, existing partitions are *reverted* to nursery state, this gives an opportunity for the cameras within them to migrate to other partitions, and also breaks apart inappropriately large partitions.

Extensive future work is required to evaluate this idea; in particular we will be interested to discover whether an initially bad partitioning converges into a good one.

9. RELATED WORK

Previously, activity topology has been learnt by tracking people as they appear and disappear from camera fields of view (FOVs) over a long period of time. For example, in [3] the delay between the disappearance of each person from one camera and their appearance in another is stored to form a set of histograms describing the transit time between each camera pair. The system is demonstrated on a network of 3 cameras, but does not scale easily as it requires that correspondences between tracks are given during the training phase when topology is learnt.

Dick et al. [4] suggest an alternate approach whereby activity topology is represented by a Markov model. This does not require correspondences, but does need to learn n^2 transition matrix elements during a training phase and so does not scale well with the number of cameras n.

Ellis et al. [5] do not require correspondences or a training phase, instead observing motion over a long period of time and accumulating appearance and disappearance information in a histogram. Instead of recording known correspondences, it records every possible disappearance that could be related to an appearance. Over time, actual transitions are reinforced and can be extracted from the histogram with a threshold. A variation on this approach is presented in [6], and has been extended by Stauffer [7] and Tieu et al. [8] to include a more rigorous definition of a transition based on statistical significance, and by Gilbert et al. [9] to incorporate a coarse to fine topology estimation. These methods rely on correctly analysing enough data to distinguish true correspondences, and have only been demonstrated on networks of less than 10 cameras.

10. CONCLUSION

The main contribution of this paper is to report on the implementation of a scalable system for automatic and on-line estimation of activity topology. Measurements of the key system component (based on the *exclusion* algorithm) indicate scalability to networks with a thousand cameras, and we report measurements on over a hundred camera streams (this limit is based on the number of cameras and computers presently available to us, not scalability). Finally, we suggest an approach for *distributed exclusion* which has the potential to scale to networks with tens of thousands of cameras.

11. REFERENCES

[1] M. Valera Espina and S. A. Velastin, "Intelligent distributed surveillance systems: A review," *IEE Proceedings - Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 192–204, April 2005.

[2] C. Stauffer and W. E. L. Grimson, "Learning patterns of activity using real-time tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 747–757, 2000.

[3] O. Javed, Z. Rasheed, K. Shafique, and M. Shah, "Tracking across multiple cameras with disjoint views," in *IEEE Int. Conf. Computer Vision*, 2003, pp. 952–957.

[4] A.R. Dick and M. J. Brooks, "A stochastic approach to tracking objects across multiple cameras," in *Proc. Australian Joint Conference on Artificial Intelligence*, 2004, pp. 160–170.

[5] T. J. Ellis, D. Makris, and J.K. Black, "Learning a multi-camera topology," in *Joint IEEE Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS)*, 2003, pp. 165–171.

[6] T. H. Ko and N. M. Berry, "On scaling distributed low-power wireless image sensors," in *Proc. 39th Annual Hawaii International Conference on System Sciences*, 2006, vol. 09, p. 235.

[7] C. Stauffer, "Learning to track objects through unobserved regions," in *IEEE Computer Society Workshop on Motion and Video Computing*, 2005, pp. II: 96–102.

[8] K. Tieu, G. Dalley, and W.E.L. Grimson, "Inference of non-overlapping camera network topology by measuring statistical dependence," in *Proc. IEEE International Conference on Computer Vision*, 2005, pp. II: 1842–1849.

[9] A. Gilbert and R. Bowden, "Tracking objects across cameras by incrementally learning inter-camera colour calibraion and patterns of activity," in *European Conference on Computer Vision*, 2006, vol. 2, pp. 125–136.