Distributed Query Processing for Mobile Surveillance

Stewart Greenhill and Svetha Venkatesh
Department of Computing, Curtin University of Technology
stewartg@cs.curtin.edu.au, svetha@cs.curtin.edu.au

ABSTRACT

Addressing core issues in mobile surveillance, we present an architecture for querying and retrieving distributed, semi-permanent multi-modal data through challenged networks with limited connectivity. The system provides a rich set of queries for spatio-temporal querying in a surveillance context, and uses the network availability to provide best quality of service. It incrementally and adaptively refines the query, using data already retrieved that exists on static platforms and on-demand data that it requests from mobile platforms. We demonstrate the system using a real surveillance system on a mobile 20 bus transport network coupled with static bus depot infrastructure. In addition, we show the robustness of the system in handling different conditions in the underlying infrastructure by running simulations on a real, but historic dataset collected in an offline manner.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Algorithms, Measurement

Keywords

mobile surveillance, distributed query processing, video indexing, virtual observer, spatial query, visibility query.

1. INTRODUCTION

Conventional architectures have involved using networks of static cameras for wide area surveillance. However, these architectures are limited in the spatial coverage that is possible. An exciting new infrastructure is a network of mobile cameras, such as that mounted on public transport networks [6]. This mobile network facilitates collection of surveillance data over a far wider area than would be possible with fixed infrastructure of sensors. However, these networks present new and challenging problems, particularly in the area of distributed querying. The problems arise because there is too much data, it is distributed in mobile and static platforms and cannot be accessed in continuous fashion.

In conventional fixed-camera networks, the connectivity between the sensors and the data center is generally fixed and predictable. In contrast, a mobile surveillance network relies on temporary connections between the sensors and the network to retrieve recorded data. Generally the bandwidth available in these connections is insufficient to retrieve all of the recorded data, so it is important that the sensor network facilitate retrieval based on demand.

Previous works on distributed querying have not dealt with the specific challenges that arise in this environment, in particular, the nature of the distributed, semi-permanent data, the lack of continuous access and the inability to have the bandwidth capacity to download all the data. Most assume the infrastructure is such that all the data can be accessed, continuously, even though it is distributed. Additionally, these systems address database type queries. Surveillance queries, on the other hand, have different intrinsic properties: the need to specify not only spatio-temporal constraints, but also the capacity to vary attributes like sampling rates in the query to match degree of interest in an area. No previous works has addressed these specific challenges in a distributed framework.

Wireless sensor networks share many issues with mobile camera surveillance: limited connectivity, limited storage and processing power. In ad-hoc sensor networks routing is an important issue, but data volumes are usually small so providing a route can be found it is usually possible to deliver the data. These networks generally do not deal with data such as video and audio.

There are also limited attempts at trying to tackle mobile networks of cameras. A notable exception is the work of [6] allowing a user to visualise a selected place at a selected time in a network of mobile sensors. However, this work is limited in two main ways: (a) The main query operators are simple and restricted to single camera, fixed sampling rate visibility queries. These methods lack the ability to specify variable sampling rates, multiple cameras and more complex queries like coverage based queries. (b) The work addressed the query/visualisation framework but did not address distributed query processing or data storage or data collection.

Thus, the open problem involves querying in a sensor network system, in which data is:

- distributed between static and mobile platforms

- semi-permanent, in that many of the storage units have to be re-used frequently
- available in parts, in that some data to answer a query may have been retrieved and thus available, whilst some data may be on the mobile device and thus needs to be retrieved on demand
- retrievable on demand, but the network prohibits complete retrieval of data.

To address these open problems and tackle the complexity of the data and the infrastructure, we need a system architecture to:

- Coordinate requests from multiple clients using a rich set of query operators
- Use available network connectivity to provide the best quality of service. This means: adaptively decide *where* and *when* queries are executed. The system needs to efficiently minimise the amount of data that needs to be transmitted between network nodes.
- Allow for query results to be adaptively refined over time. Here we need to distinguish between the potential and actual completeness of a query. For a new query, a certain amount of data may be already available in the network. In addition, it may be known that more data is available but that the user may need to wait for sensor nodes to come on-line before the data can be retrieved.

We describe an architecture for distributed querying across static and mobile platforms with multi-modal data (video and GPS). Our solution is a distributed storage system with retrieval on demand. The query drives a best effort at data retrieval. We introduce new querying operators, the *coverage* operator that can give low level of detail over large spatial areas, and generalise the observation operators to provide high level of detail over small spatial areas. We describe the query resolution process, the process of mapping spatial queries and trajectory data to temporal video segments and the subsequent distributed processing system that describes how the query resolution is performed in this distributed network. The result of the users query in the space covered by these sensors is the reconstruction of the scene at these points using observations that are either available or retrieved on demand. We show its application to a mobile 20 bus network in conjunction with a static bus depot network for surveillance and additionally, we demonstrate the properties of the system using a large set of real data collected in an offline manner, so that we can simulate the performance under differential properties of the infrastructure. The novel aspects are a system that:

- accommodates distributed static and mobile video data
- accommodates on-demand data in an adaptive fashion
- introduces new kinds of spatial-temporal queries, particularly geared towards multimedia surveillance

The significance of the work lies in the development of fundamental new paradigm for formulating the unique challenges in multimedia, sensor networks and the presentation of a solution to spatio-temporal querying in such a network, efficiently managing retrieval of available data and on-demand data.

The layout of this paper is as follows: section 3 describes the sensor network, and section 4 describes the architecture of the application. Section 5 covers two important aspects of query processing: query resolution (5.1) and distributed processing (5.2). Section 6 describes implementation of the system, and section 7 covers experiments using the system.

2. RELATED WORK

Most Internet-protocol-based network applications assume a set of underlying properties in the network: that there is an end-to-end path between a node and its peer, and that the round-trip time is not excessive. Several classes of *challenged networks* exist in which these assumptions are violated. These include sensor networks where communication must be *scheduled* to handle changes in node location, or to conserve node power. *Delay Tolerant Network* architectures (DTN) have been proposed [5] to handle communication within and between challenged networks. Typically, these involve a message forwarding system that operates above the physical network layer. A DTN architecture includes *gateways* which route messages between *regions* which might otherwise be disconnected. Gateways can include routers carried by vehicles such as commuter buses.

One application of DTN is DakNet [11], which provides non-realtime Internet connectivity to rural villages in India. Rural buses are equipped with a mobile access point (MAP) which uses a 802.11b wireless link to exchange data between an Internet-connected depot station and kiosks in villages along the bus route. When the bus comes within range of the kiosk a "session" occurs of roughly 2–3 minutes duration during which approximately 20 megabytes of data can be transferred in either direction.

Wireless sensor networks are increasingly being used for applications like environmental monitoring [9] and animal tracking [7]. These networks are typically comprised of small battery-powered devices that can communicate over short range but have limited storage and processing ability. When proximity of nodes allows, ad-hoc networks can be formed in which data is exchanged. A key problem in these networks is routing. Usually, the network includes "sink" nodes which connect to other networks. Many routing strategies exist, and routing can be improved by using knowledge of node mobility [8].

In some networks sensor nodes are fixed while in others they are mobile. Some networks include both fixed and mobile entities. The term "data MULE" refers to a entity that is able to exchange data between a sensor and an access points by virtue of its mobility [12]. By including MULEs as one tier in a sparse network, sensors are required to transmit data over a shorter range, leading to power savings.

Many of these same issues are relevant in mobile-camera surveillance. Although the nodes (eg. vehicles) are more capable than most wireless sensor network nodes they are still constrained relative to the potential demands of the system.

The advent of the "social web" has seen the development of web applications like Flickr [2] which allow the sharing of images collected by users. These become a valuable resource when indexed using tags or spatial positions. For example, Panoramio [4] allows geo-coded photographs to be

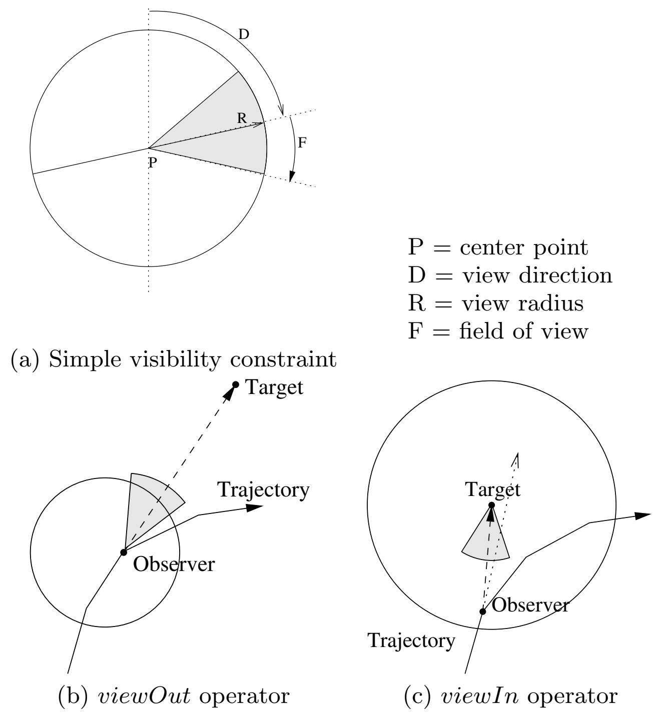


Figure 1: Interpretation of observer-target constraints for view operators.

embedded in applications like Google Maps [3]. In May 2007 Google launched StreetView, a service that allows locations to be navigated using interactive panoramas. This data is collected using panoramic cameras mounted on vehicles. At the time of writing the coverage extends only to portions of 5 US cities. Even without geo-coding, it is still possible to combine user-contributed images into scenes. Microsoft's PhotoSynth is based on work done at University of Washington [13] which combines collections of images of a place into multi-scale 3D models. Camera positions and pose are estimated from feature-point matching and can be geo-registered with respect to aerial images or digital elevation maps.

Applications like StreetView and PhotoSynth can give a powerful sense of place but lack a temporal dimension, and so cannot be used to understand events or change. An extension of the photo-sharing idea is the concept of "cyborglogging" [10], in which wearable cameras (eg. camera phones) can be used to stream images to an on-line community. This allows users to build personal narratives in real-time, and to collaborate in the observation of events. Any analysis or integration across feeds must be done by users : the system lacks an ability to navigate images in space or time.

Recent work in creating a framework for mobile surveillance, [6] uses video collected from mobile cameras to synthesise views of areas around transport routes. A "virtual observer" query allows the user to visualise the scene at a selected place over time. The system provides visibility query operators (ie. viewIn, viewOut) which return small segments of video that match the query constraints. A *simple visibility constraint* is a tuple $\langle P, R, D, F \rangle$, where P is a point in space, R is a visibility radius, D is a view direction, and F is a field of view. This is depicted in Figure 1(a). A simple visibility constraint defines an acceptance area and view range. The area is a circle of radius R centered at P. The view range is the range of directions between $D - F$ and $D + F$.

The two fundamental visibility operators are *viewOut* and *viewIn*: both use simple visibility constraints, but interpret the constraints differently as shown in Figure 1. In both cases, the observer is located inside the view area. For the *viewOut* operator (b), the view target is generally outside the defined area, although its location is unknown to the system. The angular constraint is on the direction from the observer toward the target. For the *viewIn* operator (c), the view target is the center of the defined area, and the constraint is on the direction from the target to the observer.

Whilst this query framework is useful for visualisation and querying in mobile camera applications, it does not address issues of distributed query processing, or storage or data collection.

3. SENSOR NETWORK

In a sensor network, data may be collected by mobile and static sensors, and the mobile sensors can be located spatially using sensors such as GPS. Typically, the mobile sensors have limited storage, and limited bandwidth access for data transfer. Also, in large-scale networks, the static and mobile sensors are owned by different operators, and thus the data and infrastructures are geographically separated. These operating service centres may be aggregated. Thus, the network is heterogeneous but there is no existing overarching framework across operators and sensors. For example, mobile video may be collected by different vehicle fleet operators in a city, whilst local government councils collect static camera footage. Typically, each of these operators have their own infrastructure for access, and there is no way in which this data network can be tapped in a coherent way. To complicate matters, some operators store the video whilst other do not.

Access of mobile data is a fundamental problem due to bandwidth restrictions. In a mobile surveillance network video is collected by mobile cameras which are typically mounted on vehicles. Each vehicle carries a mobile data recorder (MDR) with the capacity to store several days of data under normal operating conditions. When a vehicle returns to its depot, the mobile data recorder is interrogated via a wireless data link and video can be retrieved. A bus depot services around 100 buses, but they usually all converge around the same time, outside of "rush hours". The average operational time is 12 to 14 hours per day, which leaves about 10 to 12 hours per day for downloads. A 54Mbps 802.11g wireless link is available but in practice the effective payload throughput is about 20Mbps. At this rate, even with 100% utilisation this is only around 90Gb per day, or about 900Mb per bus. For example, 1 camera operating at 768x288 (2CIF) resolution, 5 frames per second generates about 200Kb per second, or approximately 8.6Gb data per day. Given an allocation of 900Mb per bus it is clearly not possible to retrieve more than about 10% of the generated data. This figure is obviously flexible (eg. for faster network, more cameras, etc) but it remains that it is usually not possible or desirable to retrieve and retain all data. We need a way to control data collection based on user demand.

To summarise, the network is heterogeneous, distributed across mobile and static platforms and across different oper-

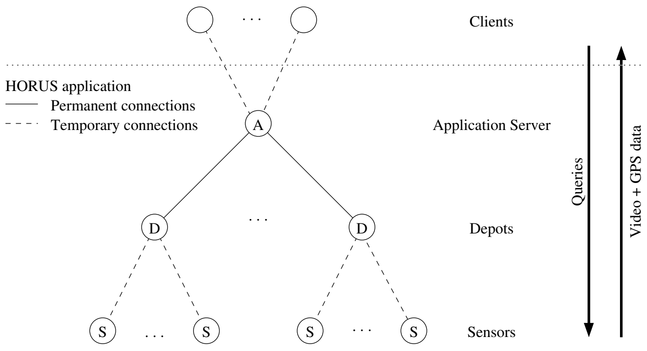


Figure 2: Sensor network consists of three tiers: application server (A-Node), depots (D-Nodes) and sensors (S-Nodes).

ators. The storage is distributed between the mobile sensors and fixed infrastructures. The communication between depots and mobile sensors is limited both in bandwidth and reliability. Therefore, when queries are made, the data must be moved in the network, and retrieved either on-demand or from existing repositories.

4. SYSTEM ARCHITECTURE

Figure 2, shows an architecture and information flow in which the sensor network implements a distributed storage and retrieval system. At the root (level 1) of the tree an application server (A-node). The application server contains a large store of video and trajectory data for use in browsing and visualisation. This connects to a number of remote service depots (level 2, D-nodes) which acquire and store video and trajectory data. The D-nodes collect information from sensor nodes (level 3, S-nodes). Connections between D- and S-nodes may be *permanent* (eg. in static camera surveillance using analog video, or video-over-fiber) or *scheduled* (eg. in mobile surveillance). In the latter case, communication can only occur when the sensor is in proximity to a wireless access point; normally this happens when the vehicle returns to its depot. In addition, the mobile data recorders are powered from the vehicle's battery so it is not possible for them to run for more than 1 or 2 hours while the engine is turned off without impacting on battery condition. The communication capacity of the D-A link is variable depending on vendor, geography, etc.

The nature of mobile storage, is that it is limited and semi-permanent. For example, a bus stores around 80Gb of data, so the vehicles at each depot have a combined capacity of approximately 8000Gb. However, data is only retained at this level of the network for approximately 4 to 6 days before the storage needs to be reused. Also, connections to these nodes are infrequent, so this data is not highly available. Therefore, the system attempts to move as much data as possible into storage within the service depot itself. This data then becomes readily available to the application server through permanent, though possibly relatively slow network links. Selected data is automatically moved from the depots to the application server for rapid retrieval by clients.

We now describe the query resolution and the distributed processing in this architecture.

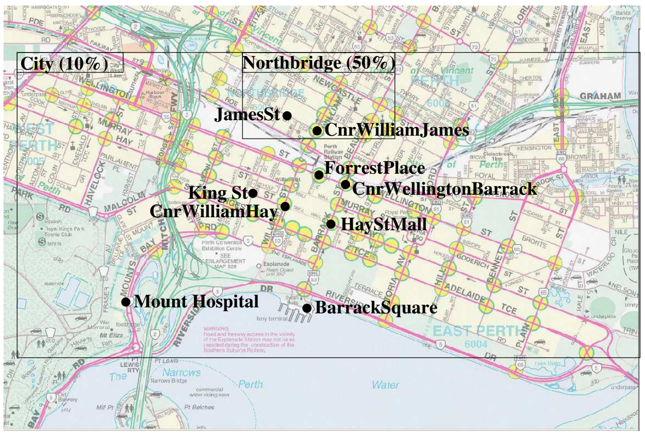


Figure 3: Example set of queries showing nine virtual observers (circles) and two coverage queries (rectangular regions). Sample rates are shown. Street directory image courtesy of Department of Land Administration, Western Australia.

5. QUERY PROCESSING

In our system, HORUS, queries are defined in a visual query environment. The user opens a map view and positions observers at places of interest. The user needs a certain level of spatio-temporal detail in order to provide context for placement of queries and to see what kind of views are likely to result. Initially, this may involve using a "tracking" query, which shows the closest observation to the mouse allowing the user to "scrub" the map to find places of interest. When placing observers it is also useful to see what data is available for different positions and orientations of the camera. Since the requested data may be off-line it is useful to be able to provide approximate (low resolution) results until the required data can be retrieved. To support this interactive browsing we need a level of background "coverage" data that is dense in space (position and orientation), but may be sparse in time. In contrast, a virtual observer query is sparse in space, but dense in time. So effectively implementing the system requires a mixture of operators that select data using different sampling criteria:

- *Observation operators*: high level of detail over a small spatial area.
- *Coverage operators*: low level of detail over a large spatial area.

Figure 3 shows a typical set of queries within the system. The "point" queries (eg. BarrackSquare, HayStMall) are *virtual observers* with radii of about 50m. In these positions we require a high level of detail; in practice we want all of the available data at the highest frame rate. The "region" queries (eg. City, Northbridge) are *coverage queries* that specify the requirements of the system for background data. Sampling rates are shown as a percentage of the full frame rate of 5fps. For Northbridge, a popular entertainment area, we require higher level of background coverage: 50 % versus 10% for the City area.

The sensor network is limited in the amount of video that can be retrieved. The aim of the system is to provide the best possible level of detail given the constraints of the sensor network. Virtual observer queries are of high value to the system, but may only require a small part of the overall network bandwidth. Coverage queries use the remaining bandwidth in a targeted way to provide background data for browsing.

Two important aspects of the problem are:

- *Query Resolution.* This is the process of determining the segments of mobile-camera video that correspond to each query in the system. This involves mapping spatial queries and trajectory data to temporal video-segment descriptions. This is described in section 5.1.
- *Distributed Processing.* This describes how query resolution is performed in a distributed sensor network and how the resulting video is made available to the client. This is described in section 5.2.

5.1 Query Resolution

The previous work [6] describes a model for query and visualisation of mobile surveillance data. However, it was limited to single camera, fixed sampling rate visibility queries. To implement the distributed queries, these operators need to be generalised to include multiple camera configurations, sampling rate constraints and coverage operators.

Formally, let V be a vehicle, let $cameras(V)$ be the set of its cameras, and $trajectory(V)$ be its trajectory. Each camera C has an orientation $orient(C)$ relative to the heading of its vehicle $vehicle(C) = V$. Each camera C also defines a video sequence $vid(C) = [\langle I_0, t_0 \rangle, ..., \langle I_{N-1}, t_{N-1} \rangle]$ of N *observations*. Each observation $\langle I, t \rangle$ defines an image I and sample time t. Using $vid(C)$, we define a function that maps a time to an observation : Let $vidt(C, t)$ be the observation $\langle I, t' \rangle \in vid(C)$ such that t' is closest to t over all observations.

A *camera track observation* is a tuple $\langle C, t \rangle$, where C is a *camera*, and t is a time. A *camera track segment* is a tuple $\langle C, t_1, t_2, A \rangle$ where t_1 and t_2 are times ($t_1 \le t_2$), and A is a sampling rate. Camera track observations and camera track segments are returned by geometric queries. Associated with each camera track observation is a unique observation (a time-stamped image) $vidt(C, t)$. Associated with each camera track segment is an observation sequence (a time-stamped video segment) $[vidt(C, t_1), ..., vidt(C, t_2)]$.

In general, a query Q is a tuple $\langle op, O, A \rangle$, where op is an *operator*, O is set of corresponding constraints, and A is a sampling rate. Each operator defines a mapping between a camera C and a set of camera track segments. For *observer queries op* is a visibility operator, and O is a *visibility constraint* (described in 2). Examples of operators are:

Definition 1 (*viewOut* : View from a place). *We define function $viewOut(C, O, A)$ to be the set of camera track segments $\langle C, t_1, t_2, A \rangle$ where $V = vehicle(C)$, O is a simple visibility constraint $\langle P, R, D, F \rangle$, and $trajectory(V, t)$ is entirely contained within the circle of radius R centered at P, and the heading at $trajectory(V, t)$ is between $D - orient(C) - F$ and $D - orient(C) + F$ for $t_1 \le t \le t_2$.*

For *coverage queries op* is simply the spatial containment operator, and O is a spatial region, generally described by a polygonal boundary.

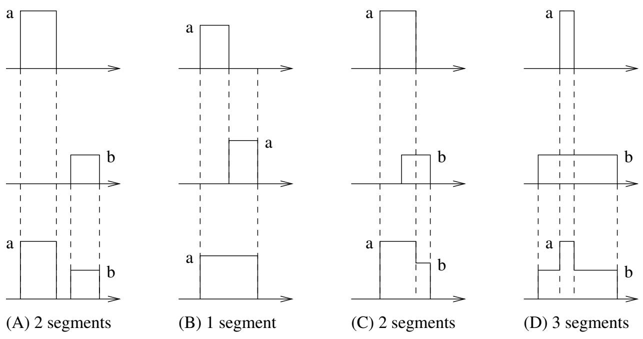


Figure 4: Example cases for merge operation. The bottom row shows the result of merging segment rates in the first and second rows. The x-axis represents time, and the y-axis shows sampling rate.

Definition 2 (*cover* : Coverage constraint). *We define the function $cover(C, O, A)$ to be the set of camera track segments $\langle C, t_1, t_2, A \rangle$ where $V = vehicle(C)$, O is a spatial region, and $trajectory(V, t)$ is entirely contained within O for $t_1 \le t \le t_2$.*

Once the trajectory $trajectory(V)$ of a vehicle is known, it is possible to resolve the video requirements for each of the attached cameras. For each query $Q = \langle op, O, A \rangle$, and each camera $C \in cameras(V)$ the system evaluates $op(C, O, A)$. The result is a sequence of camera track segments for each combination $C \times Q$ of camera and query. By appropriately factoring the internal state of each operator this computation is done using a single pass through the trajectory data. The system then merges the output for multiple queries to give a single sequence of camera track segments for each camera. This is done in a way that retains the maximum required sampling rate in the event of any overlaps. Some cases are shown in Figure 4. In case (A) two disjoint segments merge as disjoint. Where adjacent segments have equal value (B), the result is one merged segment. Cases (C) and (D) show some examples resulting from overlapping inputs; the first results in two segments, the second results in three segments. If $O = \{O_1, ..., O_N\}$ is a set of N track segment sets, we define the function $merge(O)$ to be the set of track segments forming the maximal rate over the input segments. Where O contains segments from different cameras, the cameras are merged independently.

Returning to the problem statement, given a set Q of queries we compute:

$$resolveCamera(Q, c) = merge(\{op(c, O, A) | \langle op, O, A \rangle \in Q\})$$

which for any camera c is the result of merging the output of all queries in Q for that camera. *This defines the time and sampling rate requirements for the video data that must be retrieved for camera C.* Similarly, we define

$$resolveQuery(C, q) = merge(\{op(c, O, A) | \langle op, O, A \rangle = q \wedge c \in C\}$$

which for any query q is the result of merging the output of all cameras in C for that query. *For any observation query, this defines that set of camera track segments that match the query.*

The $resolveCamera$ and $resolveQuery$ functions are used in different parts of the system (see section 7.1 for examples). $resolveQuery$ is done at A-Nodes during visualisation to collect the output of particular observer queries. $resolveCamera$ is generally run at D- or S-Nodes to determine what video must be moved into the system from sensors. This approach allows the system to minimise the amount of data to be retrieved from each vehicle, but to do this in a flexible, decentralised way that depends on specific user queries (observation queries) as well as general rules to anticipate demand (coverage queries).

5.2 Distributed Processing

As described in section 3, there is sufficient network bandwidth between D- and S-nodes to retrieve about 10% of the generated video. The system aims to make best use of available bandwidth to return requested video to the client.

Formally, we model the network as a graph. Let $N = A \cup D \cup S$ be the set of nodes, where A, D, and S are the sets of A-, D- and S-nodes respectively. Associated with each node $n \in N$ is a set of resident video corresponding to camera track segments $res(n)$ and a set $traj(n)$ of vehicle trajectories. A connection $e \in E$ between nodes a and b is represented as a tuple $\langle a, b, f \rangle$, where $f(t)$ is a *connectivity function* that express the connection bandwidth as a function of time.

Queries in the system originate from A-nodes and move down the tree. Trajectory data moves up the tree, at low cost because the volume is relatively small (1Mb per sensor per day). Video data moves up the tree, but the amount of data that can be moved is constrained by the bandwidth between nodes.

Depending on the state of the system, there are several possible relationships between a camera C and a query q. We say that q is *resolvable* with respect to C at node n if the required trajectory data is available at node n:

$$trajectory(vehicle(C)) \in traj(n)$$

We say that q is *materialised* with respect to C if q is resolvable and the result is resident at node n:

$$resolveQuery(C, q) \subset res(n)$$

The main possibilities are therefore:

- A query *unresolvable* at n if the trajectory data has not moved up the tree to node n.
- A query is *resolvable* but *unmaterialised* if the trajectory data is available, but the video data is not available.
- A query is *materialised* if both trajectory and video data is available. A query may be *partially materialised* if some video data is available but some is not available. This may occur if some coverage data exists but at a lower than required sampling rate, or if data is available at the correct rate, but for only part of the time range of a query.

Query resolution ($resolveQuery$, $resolveCamera$) can occur at any level of the tree at which the required trajectory data exists. For *interactive queries* (using $resolveQuery$) such as tracking, browsing, and placement of new observers we usually require that the queries be resolved *over all cameras* and that the results be *rapidly materialised*, so these queries execute at the top of the tree and return small amounts of data, either directly from A-node storage, or by pulling data from storage at the relevant D-node. For non-interactive *data-gathering* queries such as permanent virtual observers and coverage queries, resolution uses $resolveCamera$ at the lower levels, either at S-nodes, or at D-nodes. These queries generally need to be resolved but do not need to be rapidly materialised, and are processed with respect to the cameras on a particular vehicle when new trajectory data becomes available. Their role is to pull data from sensors into the network.

Query materialisation can occur at differing degrees at different levels of the tree. Most of the video data exists at the bottom of the tree (ie. at D-nodes). Due to bandwidth constraints on the A-D-node links, only a portion of the available data will be resident at A-nodes. Queries are generally propagated down the tree from A-nodes until they can be serviced.

While the model allows that queries be executed at S-nodes, the current implementation is constrained by the type of processing that can be done on the commercially-available mobile data recorders. In practice, data-gathering queries are resolved at D-nodes whenever buses return to their depot at the end of the day. Once the GPS data has been downloaded, the queries are resolved and the required camera track segments are requested through the vendor's existing fleet-management API. The combined requirements over all vehicles can be analysed to provide a prioritised schedule of what video needs to be retrieved and stored. High-rate video segments (from observer queries) are relatively short in duration and are easily serviced by the system. The remaining bandwidth is dedicated to retrieving coverage data for browsing. This is less critical and the associated sampling rates can be adjusted to fit the available network capacity.

6. IMPLEMENTATION

HORUS is implemented in Java, and consists of several parts: a storage manager, a query processor, and visual environment (described in [6]).

The storage manager implements storage schemes for image and trajectory data. A trajectory is stored in a single file as a stream of binary records ordered by time. This allows all or part of the trajectory to be processed by sequentially reading a section of the file. Video is stored in a container file as "blobs" of raw JPEG images. A region of the container file is an index with the time-stamp, position and size of each blob. Both trajectory and image containers are temporally sorted and accessed using either time (by binary-search of the file or index) or ordinal position.

The query processor implements the operators described in 5.1, including $viewIn$, $viewOut$ and $cover$. The outputs from multiple queries are merged to produce compact camera track segment lists. These are used in two ways. Firstly camera track segments are used to build time-lines [6] for navigating the video data in response to interactive queries. Secondly camera track segments define segments of video that are to be imported into the system for standing queries.

Figure 5 shows how HORUS currently integrates with existing system at a bus operator's depot. Each bus contains a recorder which stores GPS and video data in a proprietary file format (DVSS [1]). A *fleet management system* manages the retrieval of data from the bus fleet. A stand-

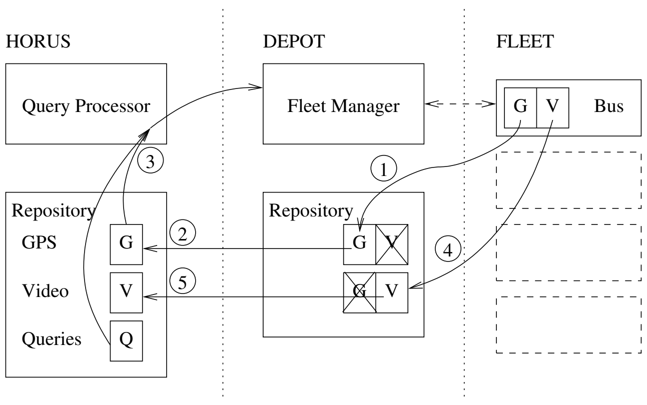


Figure 5: HORUS implementation at bus depot.

ing request is placed in the system for all new GPS data. As buses come on-line GPS data is retrieved into the depot repository (step 1). A HORUS process monitors the depot repository for new data and imports this into its own repository (step 2). All new trajectory data is then processed against all standing queries to produce a stream of camera track segments which is submitted as a batch of requests to the fleet management system (step 3). Later, the image data is retrieved to the depot repository (step 4) and is then imported into the HORUS repository (step 5). The import process may filter out information that is present in the depot repository (eg. to select particular channels, or resample data). The HORUS repository is independent of vendor-specific data formats and protocols, although it does use these in the process of gathering data.

7. EXPERIMENTS AND DISCUSSION

We evaluated the design of HORUS using a number of data sets.

First, we had access to the live output for about 20 buses at a depot via the system shown in Figure 5. These buses form the Central Area Transit (CAT) service, a free high-frequency service in the Perth central business district. This data (named "CATLIVE") was collected on-line with a latency of about 1 day, but was restricted as detailed below.

The second data set (named "CATVID") consists of 1.2TB of video and trajectory data, and represents the output for the 20 buses over about 4 days. This was collected directly from the bus recorders. In addition, in a third data set ("CATGPS"), we used historical GPS data for these same buses lasting up to 9 months into the past. Only the last few days of the GPS tracks overlaps available video.

The CATGPS data set was used to evaluate some aspects of the design that could not be tested because of limitations in the existing proprietary depot infrastructure. In particular, the system lacked an efficient scheduler so downloads were limited to time-slots of fixed length. It was not possible to "wake up" buses at predetermined times so the time-window available for downloads was unreasonably small. Also, the system did not allow selected video channels to be extracted, and it was not possible to temporally resample the video. These factors limited the volume of video that could be retrieved into the CATLIVE set. Therefore, to study network throughput of the system, we used the CATGPS data to "simulate" the effects of variable bandwidths and sampling rates.

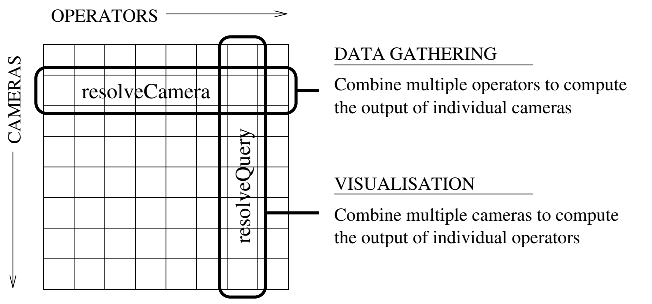

OPERATORS
CAMERAS
resolveCamera
resolveQuery
DATA GATHERING
Combine multiple operators to compute the output of individual cameras
VISUALISATION
Combine multiple cameras to compute the output of individual operators

Figure 6: Query resolution merges outputs of camera-operators in different ways depending on application.

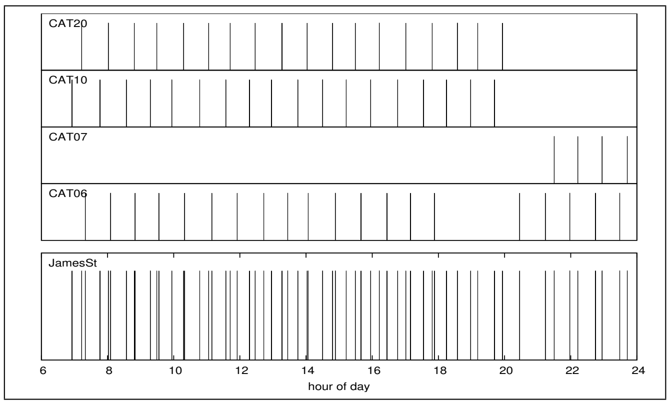


Figure 7: Output for one observation operator (bottom plot) is made up of the contributions of four mobile cameras (top plots).

7.1 Query Resolution

This section demonstrates the query resolution process using the 20 CAT buses accessible through the bus depot. Additionally, the coverage of the system was validated extensively on the CATVID dataset, but these results are not presented here due to space restrictions. Instead, we present the results using the CATLIVE data with the configuration depicted in Figure 5 and using the queries shown in Figure 3. We demonstrate two applications of the resolution process within HORUS : the computation of results for individual observation queries, and the computation of results for individual cameras. As shown in Figure 6 we can consider each combination of camera and query $C \times Q$ to be elements in a matrix of camera track-segment results which are merged in different ways. For *visualisation*, *resolveQuery* merges multiple cameras to derive the output of individual operators. For *data gathering*, *resolveCamera* merges multiple operators to derive the output of individual cameras.

7.1.1 *Resolution of Individual Queries (resolveQuery)*

Function *resolveQuery* computes the results of individual operators (eg. an observer query) by combining the output of multiple cameras over time. This is used by the visuali-

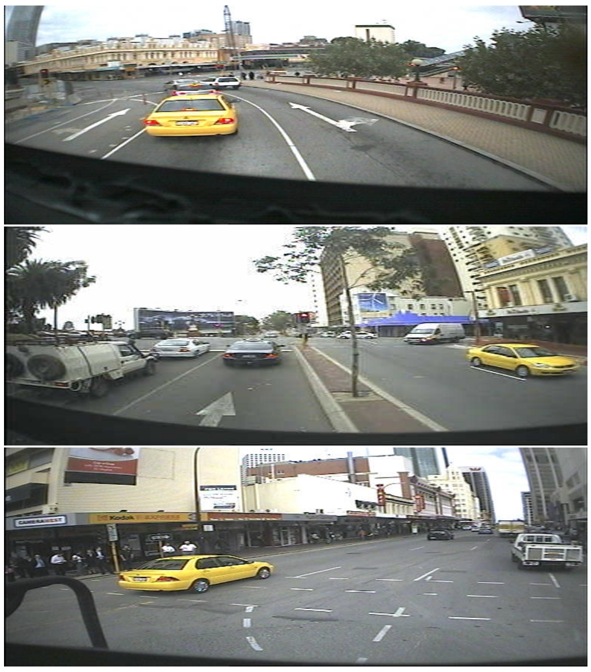


Figure 8: Observations of a moving object (yellow car) gathered from mobile cameras.

sation system to generate views of a place. The process is illustrated in Figure 7. Here, four different vehicle cameras contribute observations of a place over the course of a day. The horizontal axis shows time of day, and the vertical axis shows sampling rate. Each "spike" in the plot corresponds to a camera track segment of high sampling rate. The bottom trace ("James St") shows the operator output which is a combination of the outputs of the individual cameras.

Figure 9 shows the materialised output of this query at different times of the day. The images depict a day in the life of a Northbridge restaurant, showing changes in appearance between morning and evening (see the caption for details).

The previous output was generated using a *viewOut* operator, which find views from a point in space. When the location of an object of interest is known we can use the *viewIn* operator to find observations of that object from different perspectives (see [6] for details). This applies to both stationary objects and to moving objects whose trajectory is known.

In this experiment we drove a distinctive yellow car around the city while carrying a hand-held GPS unit. We placed a virtual observer on the vehicle by linking the location of a *viewIn* query to its GPS trajectory. The result was a set of roughly 50 observations of the vehicle over a period of a few hours collected from the bus fleet. Figure 8 shows some examples. This technique does not deal with problems like occlusion, but gives a workable set of candidates that can be easily filtered interactively.

7.1.2 Resolution of Individual Cameras ($resolveCamera$)

A second application of query resolution is to drive the collection of video data from the sensor network, that is, the data gathering process. Here, function $resolveCamera$ computes the output of an individual camera by merging the contributions of multiple operators.

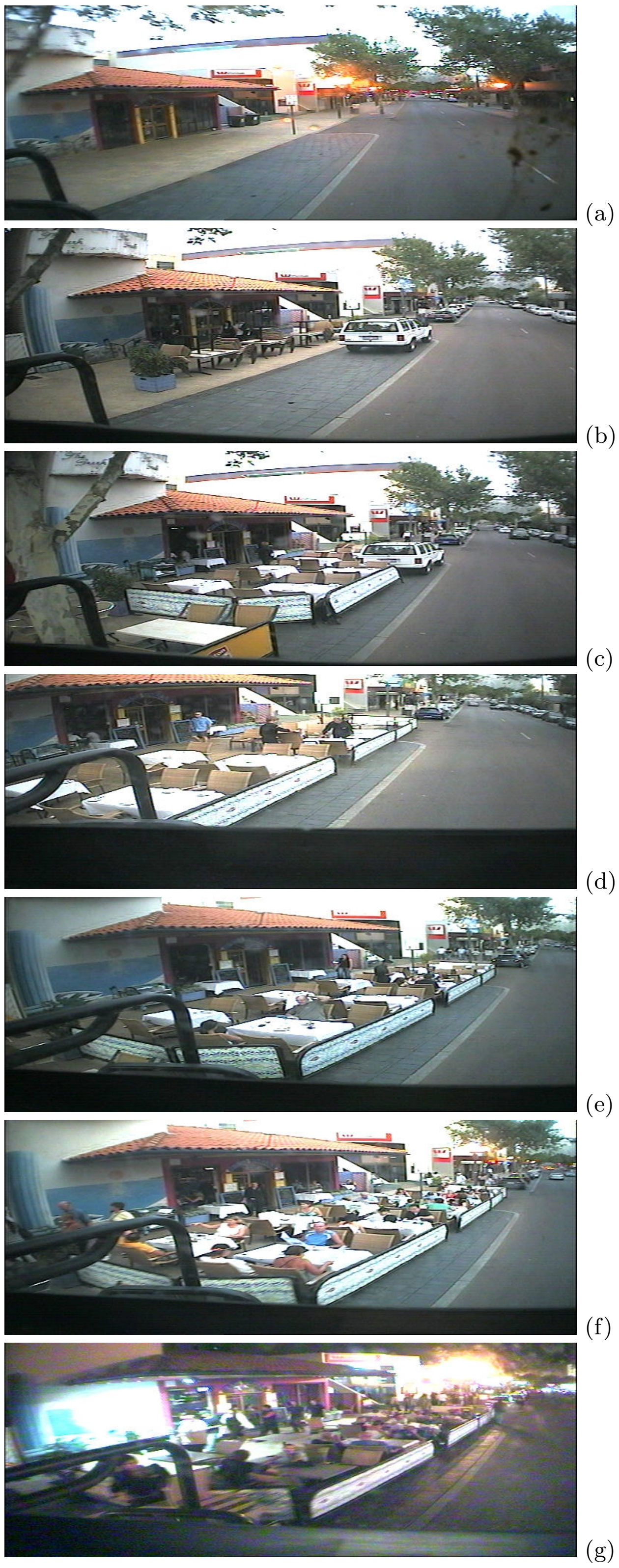
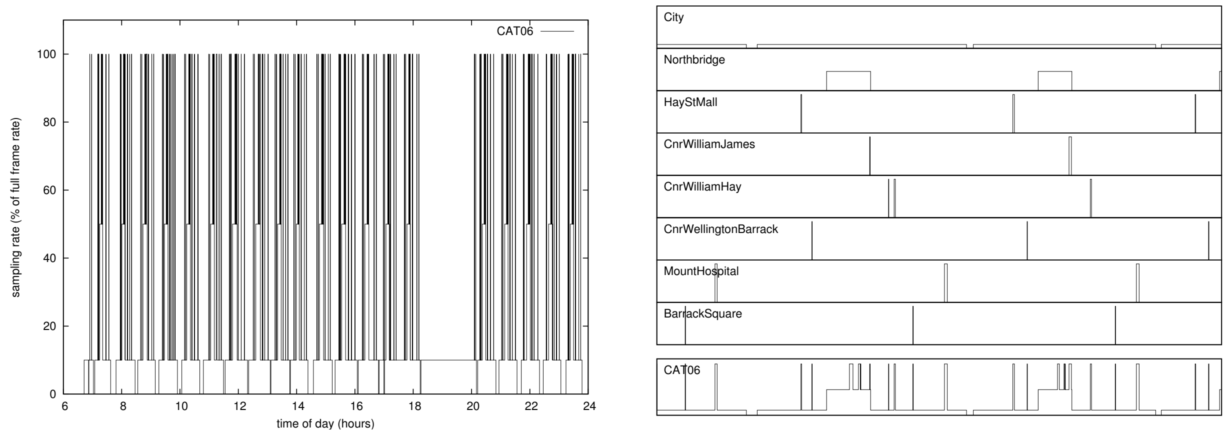


Figure 9: Time-lapse view showing a day in the life of a Northbridge restaurant. Early morning: the streets are deserted (a). Later (b), tables are stacked, waiting to be laid out for alfresco dining. Initially, part of the dining area is blocked by a parked car (c). Later, the seating is ready, and waiters are setting the tables (d). Patrons begin to arrive (e), and stay late into the evening (f–g).



Full Day 2 hour segment (12:00 to 14:00)

Figure 10: Video requirements for one camera, CAT06, traversing the "Blue CAT" route on 19/1/2007. These results are for the queries shown in Figure 3 which includes seven virtual observers and two coverage regions. The left plot shows the requirements over as 24 hour period. The right plot shows detail over a two hour period, illustrating contribution of individual operators (top traces) to the final result (bottom trace).

This is demonstrated in Figure 10, which shows a portion of the requirements for a bus on the "Blue CAT" route. The bus traverses a circuit 21 times over the period of a day. This circuit is covered by nine virtual observers and two coverage queries (see Figure 3). The background rate is defined by coverage queries: the City query is 10%, and the Northbridge query is 50%. The horizontal axis shows the time of day. The vertical axis shows the requested video sampling rate as a percentage of the full frame rate. The left plot shows sampling rate over a 24 hour period. The right plot shows detail over a two hour period, and indicates how *resolveCamera* computes the resultant signal (bottom trace) by merging the set of outputs for specific queries (top traces). The observer queries are generally disjoint in time, being based on non-overlapping target regions. The coverage queries overlap the observer queries as well as other coverage queries. Each time the bus passes through a virtual observer a short segment of high-resolution video is required, resulting in a "spike" in the graph. The "step" patterns around 12:30 and 13:30 correspond to a transition between coverage operators where the background rate changes between 10% (City) and 50% (Northbridge).

7.2 Network Simulation

HORUS relies on network connections to propagate video upstream from S- to D- and A-Nodes. Data that is closer to the A-Nodes can be retrieved with lower latency, so we try to maximise the amount of data moved up the network tree. Queries are used to prioritise which data is retrieved into the system, but the system is ultimately limited by available network bandwidth. To explore these factors, we simulated the performance of the system on real data (CATGPS dataset) with links of varying capacity.

The simulation is determined by a number of parameters (default values in parentheses). We assume that each day a time window t_{down} (10 hours) is available for downloading data from the sensors, and that the sensors can be scheduled to be on-line at the relevant time during this window. The available bandwidth for downloads is bw_{down} (20 Mbit/S). Based on 2CIF video at 5fps stored as JPEG, the data rate for one camera (*videoRate*) is approximately 200Kbytes per second, or around 1.6Mbits per second. The amount of video that can be downloaded per day from S- to D-Node is therefore $t_{down} \times bw_{down}/videoRate$ or approximately 125 hours per day at full frame-rate. Similarly, t_{up} (24 hours) and bw_{up} (1 Mbit/s) define the upload link capacity. These values are typical of an ADSL-2 link, which can handle approximately 15 hours video per day.

The simulation uses GPS and query data to derive a list of camera track segments to be downloaded each day from S-Node S_i. These are added to a $pending_{down}$ list which is prioritised by sorting by sample-rate and then start-time. The available download time is distributed amongst these segments, in order of priority. The downloaded segments are moved from the pending list to two lists: a $res(D)$ list of resident segments at the D-Node, and a $pending_{up}$ list of segments waiting to be uploaded. The same process is repeated for the up-link: pending uploads are prioritised and transferred to a list $res(A)$ according to link capacity.

Figure 11 shows simulation results for 1Mbit/s (left) and 250kbit/s (right) links between A- and D-Nodes; other parameters are the values described above. The result includes three plots. The top plot shows the number of vehicles on-line (varies from 5 to 18), and the amount of video offloaded from the sensors. The times are hours of full-frame-rate equivalent video, so 1 hour at 10% sampling rate is treated as 6 minutes. The test data spans a period of about 300 days during which the buses were being fitted with new data recorders. Therefore the number of vehicles on-line increases over time. Note that the amount of video generated increases over time but also fluctuates on a 7-day cycle because the fleet is less active on weekends. Given these parameters, the system is able to retrieve all data from S- to D-Nodes using only about 20% of the network capacity.

The middle plot shows the amount of video (in hours) resident at D-nodes for both observation queries (bottom trace) and total over all queries (observation + coverage, upper trace). The middle trace shows the total video resi-

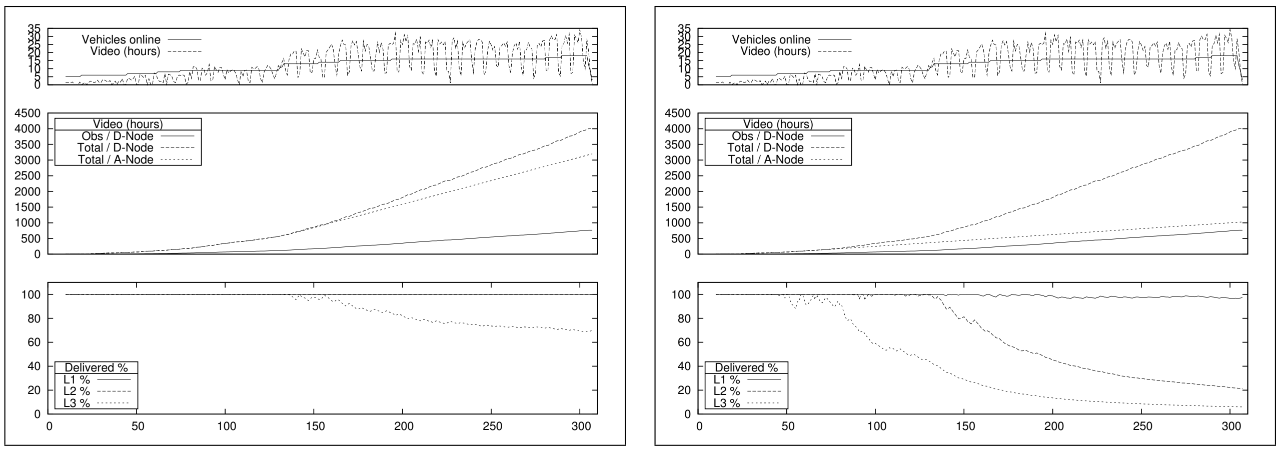


Figure 11: Network simulation results for 1Mbit/S (left) and 250Kbit/s up-links between D- and A-Nodes.

dent at A-Nodes. Notice that the "total" plot diverges as the system sheds coverage data in favour of observation data.

The bottom plot shows the percentage of the total available video that is delivered upstream from D-Nodes to A-Nodes. This is broken down by sampling-rate class: L1 is the highest priority level (observation queries). L2 is the "50% rate" coverage query ("Northbridge" on the map), and L3 is the "10% rate" coverage query ("City" on the map).

In the 1Mbit/s scenario, as the load on the system increases, the system drops the amount of coverage (L3) data to a level of about 70%, but still retains L1 and L2 data. In the 250kbit/s scenario the system is able to transfer less than 4 hours of up to 30 hours video that is generated per day. Initially (around day 50) the system begins dropping L3 coverage data, and later (day 130) starts to drop L2 coverage data. Eventually, the system is able to transfer only 6% of the L3 data, and 21% of the L2 data, but it still retains 97% of the L1 observation data.

Although the levels of coverage data vary, in both cases the system manages to successfully transmit most of the observation data upstream from D-node to A-node storage.

7.3 Discussion

Results in this section demonstrate the resolution of individual cameras and individual queries using data from a live mobile camera network. Simulations using a 9-month historical data set show how the system responds under various network conditions. In challenged environments, the system continues to deliver high-priority observation data by adaptively discarding the lower-priority coverage data. In this scenario, the affected coverage data would be retained at the D-Node, but would be accessed with greater latency in an interactive scenario. Importantly, the data is retrieved from the sensors before the associated storage is reclaimed by the mobile data recorders.

We found some limitations in the proprietary depot infrastructure but these are expected to be removed by the vendors in the near future, allowing a full-scale testing of the proposed approach. From the user's point of view, the response time of the system to interactive queries is important. This depends on the location of the required data in the network, and is difficult to characterise without reference to a particular set of new and existing queries. This is a subject for future work.

8. CONCLUSION

In this paper we have presented a way in which we can solve the core issues in mobile surveillance: distributed data on mobile and static platforms, bandwidth-limited and scheduled connections, and complex spatio-temporal queries. We have demonstrated the system on a real bus and depot infrastructure and shown the usefulness of this solution in tackling the challenging problems in this domain. In addition, we simulate varying infrastructure conditions using a historic real dataset and demonstrate the robustness of our system to handle these conditions.

9. REFERENCES

[1] Digital Technology International. Web site visited April 2006. `http://www.dti.com.au/`.

[2] Flickr. `http://www.flickr.com/`.

[3] Google maps. `http://maps.google.com/`.

[4] Panoramio. `http://www.panoramio.com/`.

[5] K. Fall. A delay tolerant network architecture for challenged internets. In *ACM SIGCOMM 2003, 25-29 August, Karlsruhe, Germany*, 2003.

[6] Stewart Greenhill and Svetha Venkatesh. Virtual observers in a mobile surveillance system. In *ACM Multimedia 2006, 23-27 October, Santa Barbara, USA*, 2006.

[7] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *ASPLOS, San Jose, CA*, October 2002.

[8] J. Leguay, T. Friedman, and V. Conan. Evaluating mobility pattern space routing, 2006.

[9] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA'02, Atlanta, Georgia*, 2002.

[10] S Mann, J Fung, and Raymond Lo. Cyborglogging with camera phones : Steps toward equiveillance. In *ACM Multimedia 2006, 23-27 October, Santa Barbara, USA*, 2006.

[11] A Pentland, R Fletcher, and A Hasson. DakNet: Rethinking connectivity in developing nations. *IEEE Computer*, pages 78 – 83, January 2004.

[12] R. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a three-tier architecture for sparse sensor networks. In *IEEE SNPA Workshop*, 2003.

[13] N Snavely, S M Seitz, and R Szeliski. Photo tourism: Exploring photo collections in 3d. *ACM Transactions on Graphics (SIGGRAPH Proceedings)*, 25(3):835–846, 2006.