Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

A Demonstration of a Flexible Network Architecture

Zhixue Wu and Feng Huang

Abstract

In order to test and demonstrate some key ideas of the FlxeiNet project, a demonstrator of a surveillance camera system has been implemented. This document describes the design and implementation of the demonstrator. At first it introduces several key technologies for implementing a flexible network architecture: switchlet, open signaling, open switching, virtual network, QoS based routing, and group-based connection control,. Then it describes how these technologies are used in the demonstrator.

Although it is impossible to get the performance statistics from the demonstrator because of the nature of simulation, the flexibility of the architecture has been clearly shown.

APM.2013.00.03 **Draft** 30th June 1997

Technical Report

Distribution:
Supersedes:
Superseded by:

A Demonstration of a Flexible Network Architecture

A Demonstration of a Flexible Network Architecture

Zhixue Wu and Feng Huang

APM.2013.00.03

30th June 1997

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by APM Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

APM Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 359779
E-MAIL	apm@ansa.co.uk

Contents

1 Introduction

There exist several problems with today's networks [Tennenhouse97]: the difficulty of integrating new technologies and standards into the shared network infrastructure, poor performance due to redundant operations at several protocol layers, and the difficulty of accommodating new services in the existing architectural model. As a result, developers of a distributed application have to fully understand the environment in which the application is to be deployed. This makes distributed application development complex and time-consuming. Also, the developed application is hard to evolve when the environment changes. On the other hand, the emergence of mobile code technologies make dynamic network service innovation and deployment attainable. FlexiNet intends to address this issue by separating application development from its deployment and providing support for flexible and dynamic application deployment.

In order to investigate the application scenarios and to produce useful inputs to the FlexiNet project, we have implemented a surveillance camera simulation system as a demonstrator. This report presents its design and implementation.

1.1 Market and technological development trends

Current market and technological development trends in broadband networks development require a flexible network architecture:

- A new market for the distribution of a variety of services over a single network is emerging. This is because the decrease of cost and the increase of transmission capacity would make the domestic broadband access over fixed and wireless networks become commercially feasible soon. Some typical new services are Internet connectivity to basic telephone service and video-on-demand service.
- Heterogeneous networking in terms of services, protocols, control functions and devices is a result of the above trend. A very flexible network architecture is essential for supporting heterogeneous networks, such as a combination of a digital multimedia distribution network and a global Internet-like packet-switched network.
- The demand to new services also triggers requirement for rapid network innovation process. Currently, from prototype to large scale deployment takes about 10 years. To improve this situation, a network architecture must be scalable and allow new services to be deployed easily in an existed network.
- Different types of service would pose different requirements for quality of service (QoS) to the multi-service network. For example, a multimedia application has much stricter time constraints than a file transfer application.

1.2 Requirements of a flexible network architecture

To resolve the problems of today's networks and to meet the challenges from the current trends, a flexible network architecture should provides the following properties:

- A sophisticated, conceptually simple architecture
- Support for heterogeneous networking
- Easy service creation and deployment
- Support for Quality of Service

1.3 Approach

Some researches [Edwards96,Tennenhouse97] have been done on providing flexible network architectures. A basic approach has been taken is to provide an open interface to network elements, such as switches, routers and nodes. Open switching is based on the idea of separating hardware switching functions from the call and connection control functions. It is an important factor in creating a flexible network architecture, because by implementing the control functions independently from the physical branching, routing or switching equipment, the same hardware can easily be used in different control architectures. On the other hand, it also enables service suppliers to define a control policy appropriate to their services.

2 Key Technologies

In this section, we introduce some key technologies that are important and useful for implementing a flexible network architecture.

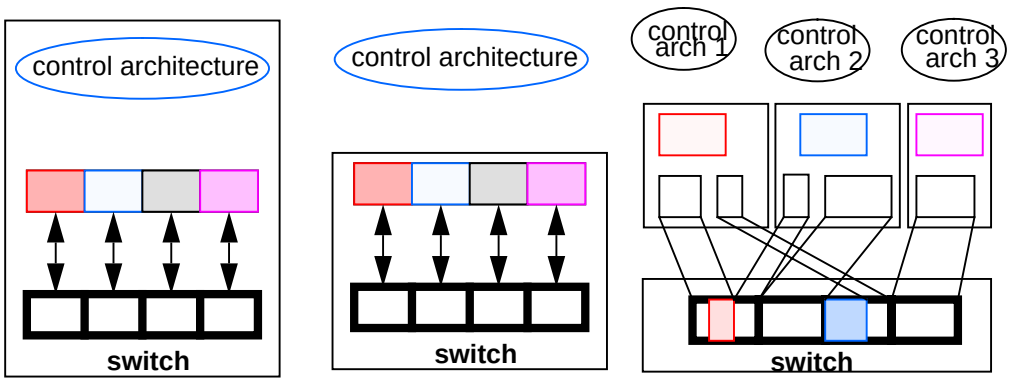
2.1 Switchlets and Virtual Network

The key requirement for a flexible network architecture is that it must be capable of supporting all existing and future services. It is clearly impossible to try and define all future services. This means that signalling protocols will constantly need to be modified as new services are introduced and network users will be frustrated in their desire to implement their innovative services. Currently the control policy is typically implemented in software running on the physical switch. This makes it impossible for service suppliers to define a control policy appropriate to their services.

To solve the above problem, one trend in broadband networking is to separate the call and connection control functions from the hardware routing and switching functions. This is important for creating a flexible network infrastructure since the same hardware can easily be used for different services by implementing the control functions independently from the physical switch.

An underlying and related problem is that of the control interface provided by the physical switch. This interface must be open so that different control architectures can be developed to make use of it.

Figure 2.1: Evolution of switch

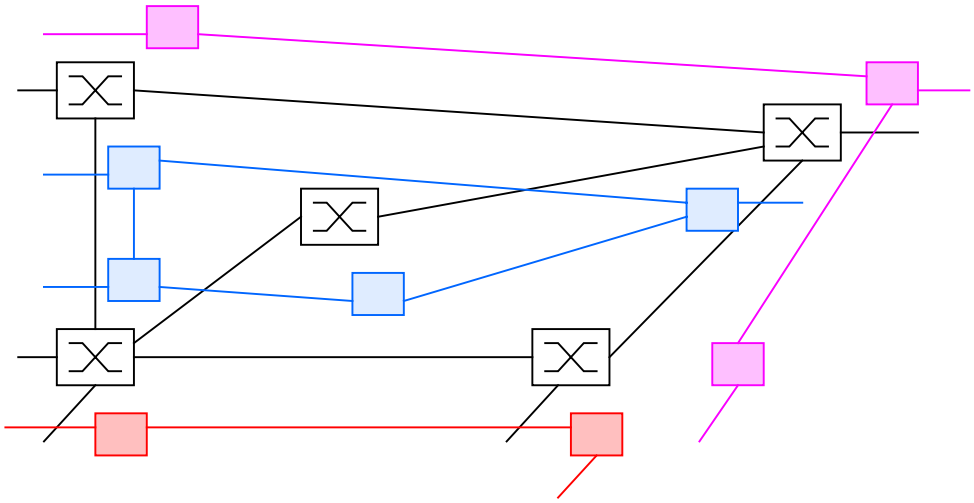


One approach to the open switch control problem is to define a simple low level interface which can be used within any control architecture to exercise control of the physical switch. This approach, followed within the DCAN project [Herbert94], has lead to the design of an open switch control interface. But it

still has the limitation that only one control architecture can be operational at any particular moment of time.

This problem can be resolved through the switchlet concept [Merwe97]. A switchlet encapsulates a subset of the physical switch resources with a particular control architecture. Since the interface of a switchlet is the same as the open switch control interface, its control architecture is not aware of that it is not in control of the whole physical switch.

Figure 2.2: Switchlet based virtual network



Switchlets are combined into virtual ATM networks, each of which can potentially use a different control architecture, *i.e.*, be of a different type. This means that switchlets permit the introduction of new control architectures into an existing network in a very elegant and controlled manner. Figure 2.2 illustrates that three different virtual networks are deployed at a physical network at the same time.

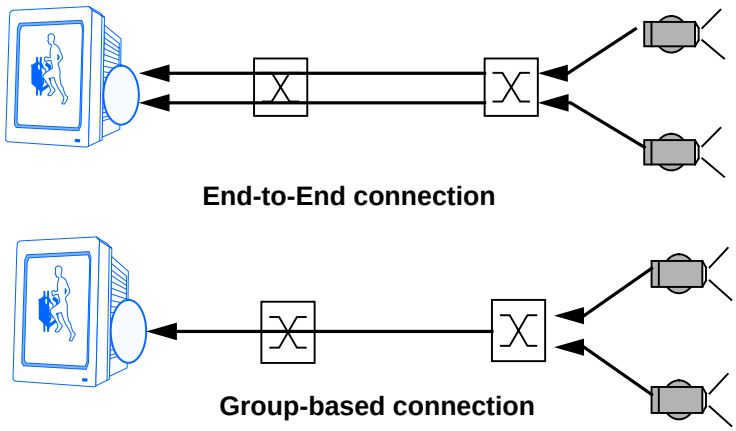
2.2 Application-Specific Connection Control

Usually, a number of logically associated connections are grouped together into a higher conceptual entity, [Rooney97] terms it as a *Call*. Applications are free to associate any groups of connections into a call. What makes the concept of call powerful is by allowing an application to create the control behaviour that is to be used within it. The association of a group of connections with the control behaviour to manage those connections is termed as a *Connection Closure* by [Rooney97]. Using connection closures, applications can take advantage of their high level knowledge about how connections are to be used within a service in order to optimise the use of their resources.

Take a security guard camera system as an example. A security guard monitors video from two different rooms each with their own camera. Suppose that the rooms are not far away from each other and that they are connected to the same switch. Based on the end-to-end connection policy, it needs to establish two connections from the cameras to the display of the security guard. However, the monitor will only ever display one camera at a time. Therefore, at any given moment one of the connections is redundant. Knowing this application-specific information, we could build a connection closure which contains a connection from each camera to the monitor and which

multiplexes the two connections, say, every 3 seconds. The connection closure consists of one connection to the monitor from the camera connected switch and periodically interchanges the input of the switch.

Figure 2.3: The security guard system



2.3 Routing Subject to Quality of Service Constraints

With new network services, applications request for not only a service, but also a service with a certain quality. Therefore, modern communication networks must cater to users with subjective QoS requirements [Hanssen97].

To meet user's QoS requirements, a system must provide enough resources to the user requested service. In a connection-oriented communication network, the transfer of information between two end users is accomplished by network functions that select and allocate network resources along an acceptable path. Routing is a network control mechanism through which a path is derived for establishing communication between a source and a destination in a network. Therefore, routing plays a very important role to meet the QoS requirements.

Path selection within routing is typically formulated as a shortest path optimisation problem, *i.e.*, determine a series of network links connecting the source and destination such that a particular objective function is minimized. The objective function may be the number of hops, cost, delay, or some other metric that corresponds to a numeric sum of the individual link parameters along the selected path. Efficient algorithms for computing shortest paths have been used in communication networks. However, within the context of satisfying diverse QoS requirements, the computation becomes more difficult as constraints are introduced in the optimisation problem. These constraints typically fall into two categories: link constraints and path constraints. A link constraint is a restriction on the use of links for path selection, *e.g.*, available capacity on a link must be greater than or equal to that required by the call. Link constraints are relatively straightforward, as one simply removes links from the topology for shortest path computation that do not meet the link selection criteria. A path constraint is a bound on the combined value of a performance metric along a selected path (e.g. end-to-end delay through the network must not exceed what the call can tolerate).

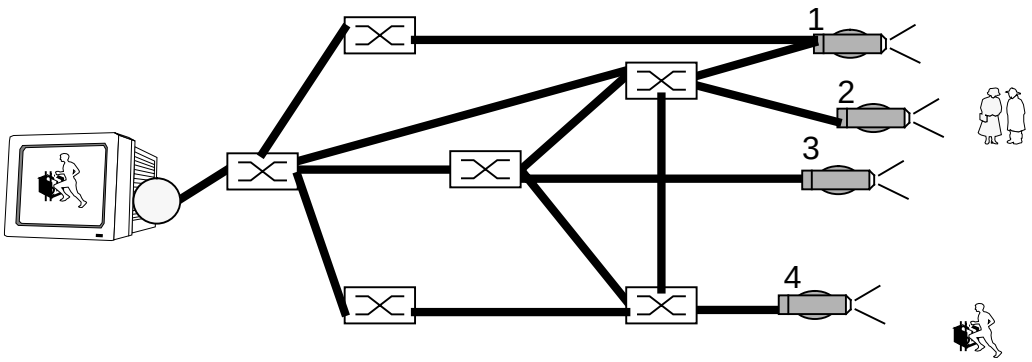
3 A Surveillance Camera System

We intend to test and demonstrate the new technologies aiming for flexible network architectures through the simulation of a wide-area surveillance camera system.

3.1 The scenario

The surveillance camera system, as shown in figure 3.1, consists of a number of cameras distributed in a wide area and a monitor. At any moment of time, the monitor displays video from one of the cameras. The cameras and the monitor are connected via switches over a wide area network.

Figure 3.1: A surveillance camera system.



Normally, the monitor rotates through the cameras in a given interval. However, the operator should be allowed to take control and choose a camera to be displayed. For example, when he finds some suspicious event from a camera, the operator might like to follow that camera so that the event can be watched more carefully.

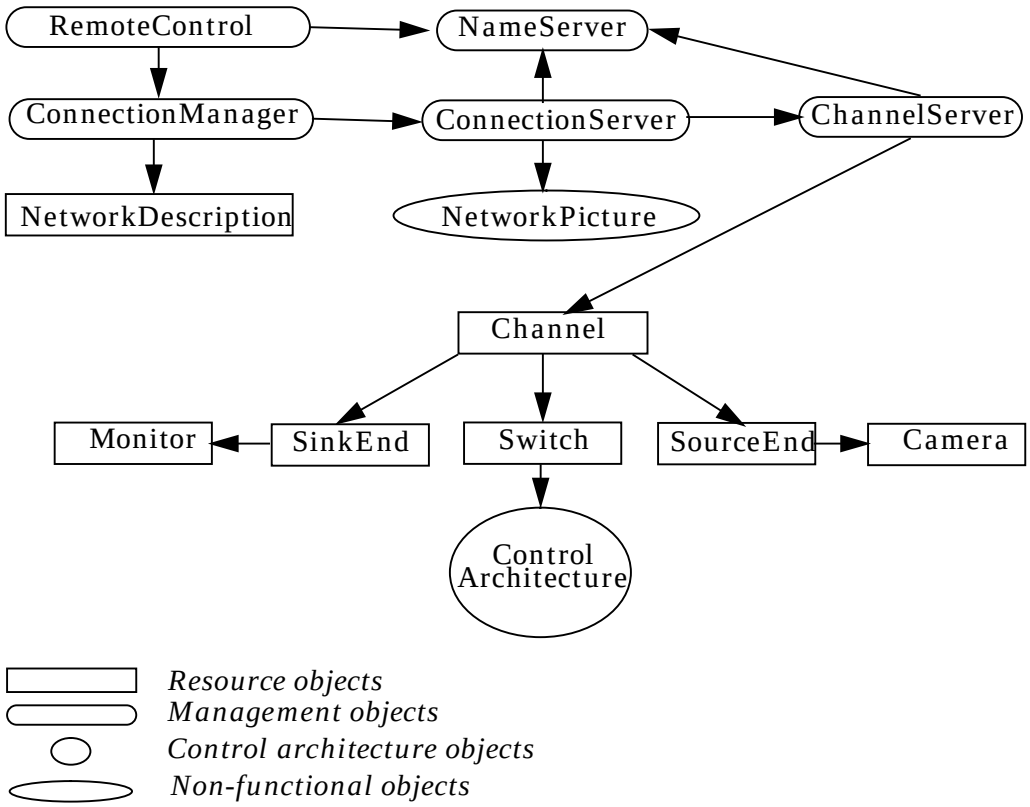
Moreover, the operator should also be allowed to decide the required quality of service. For example, in the day time, the network traffic is heavy and the occurring chance of unlawful events is small. A low quality of service is a suitable choice. During the night, the network traffic is light and there is a larger chance of criminals. It is better to request a high quality of service. Therefore, the system should be flexible on providing different level of quality of service.

3.2 The architecture

The core design of the demonstrator is based on the switchlet concept so that different control architectures can be deployed and tested. There are four groups of objects: resource objects, management objects, control architecture

objects and objects implementing non-functional requirements. The relationship between these objects is shown in figure 3.2.

Figure 3.2: The architecture of the demonstrator



Resource objects are objects used to simulate resources in the network such as switches and cameras. Resource objects include *Switch, Camera, Monitor, Channel, SinkEnd* and *SourceEnd*. A *SinkEnd* object is a special switch whose output is connected to a *Monitor* object. A *SourceEnd* object is a special switch with a *Camera* object as its input. A *Channel* object represents a connection between two switches. The *NetworkDescription* object records the representation of the network topology and the characteristics of the network elements such as the bandwidth of each channel. These information will be used by the *ConnectionManager.*

Management objects are used directly or indirectly to control the network, *e.g.*, for setting up or tearing down a connection. The *NameServer* object is an information repository which stores the information about all the network resources and answers queries about them. The *ConnectionManager* object is responsible for application-specific connection management and QoS based routing which satisfies users' requirements. The *ConnectionServer* object is responsible for setting up and tearing down a particular connection according to the *ConnectionManager*'s instructions. The *ChannelServer* object is responsible for connecting or disconnecting a channel between two switches. There is also a *RemoteControl* object for providing a graphical interface for the end users to send their requests.

The objects in the control architecture group are used to implement a particular control policy. We implemented two control policies in the demonstration, namely a manual control policy and an automatic rotation policy.

In order to demonstrate the technologies clearly, it is necessary to make the topology of the network and the dynamic connection changes over the network visible. This is the purpose of the non-functional objects, such as the *NetworkPicture* object.

3.3 Simulation

Each network element is simulated in an independent process. They communicate with each other through sockets. Each of them has a thread dedicated to accepting requests from others through a particular port called *the control port*.

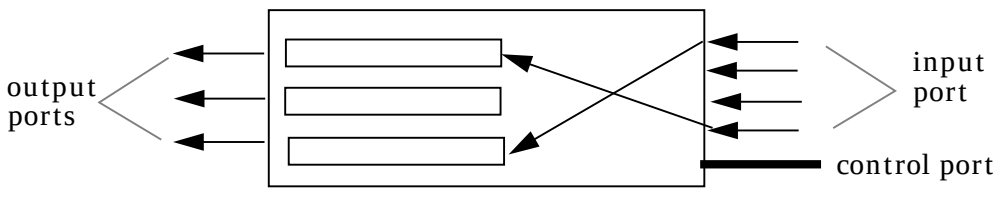
Camera A camera is simulated as a Java applet which displays a still background image to represent the scene of an inspected area. There is Duke bouncing continuously over the background. Duke's movement is decided by a random number generator with each number representing a particular step. The generator produces random numbers in a given speed.

Data transfer To simplify the implementation but without losing the behaviour of the data transfer over a network, instead of passing a series of frames of images from a camera to the monitor, we use an encoded integer to represent a frame. The integer includes a background identifier and Duke's position so that the monitor can reproduce the original image from its local image base by decoding such an integer.

Monitor Similar to a camera, a monitor is simulated by a Java applet with a still background image and bouncing Duke. Different from a camera, however, the background image displayed in the monitor and the step of Duke's movement are decided by the encoded data received from the input channel. Moreover, the interval between each step of Duke's movement is affected by the network traffic.

Switch As shown in figure 3.3, each switch has a number of input ports and a number of output ports. There is a data buffer associated with each output port. For each active input port (*i.e.*, when it is connected to a channel), a thread called *switchInThread* is created. The thread will accept any data passed in through that port, and put it in the data buffer of an appropriate output port. For each active output port, there is a thread called *swichOutThread* associated with it. This thread sends data in its buffer to the next hop through the output port.

Figure 3.3: Switch.



Channel bandwidth We take two measures to demonstrate the behaviour of channels with different bandwidth. First, the speed of delivering data out from a data buffer is partly decided by the bandwidth of the related channel. The higher the bandwidth, the faster the delivering speed. If a data buffer of an output port is overflowed due to the slow delivering speed, the oldest data in

the buffer would be dropped. As a result, a too low bandwidth channel would cause some data loss.

Secondly, to make the difference between high and low bandwidth channel more obvious, we use different quality of images to represent a background that is passed through channels with different bandwidth. To implement this, we include a quality indicator in the encoded integer representing an image. Before forwarding an encoded integer, a switch first checks whether the bandwidth of the outgoing channel is lower than the incoming one. If it is, the switch changes the quality indicator of the integer to the lower one. In this way, when the data item reaches the monitor, the quality indicator represents the final quality of the image. The monitor will display the background with an appropriate quality regarding to the quality indicator.

Control architecture We simulate two kinds of control architecture in our demonstration. The first one is a manual control policy. In this control architecture, each active incoming channel is switched to an outgoing channel. The speed of data delivering is decided by the bandwidth of the outgoing channel. The switch will continue to forward data from the incoming channel to the same outgoing channel until the connection is dismissed. Under this control policy, the operator need to switch from one camera to another camera manually through the remote control panel.

The second control architecture we implemented is an automatic rotation policy. In this case, an outgoing channel of a switch may be corresponding to several incoming channels. The switch would connect the outgoing channel from one of the interested incoming channels for a given period of time, and then change the connection to the next incoming channel, and so on. In this way, the monitor will automatically display videos from different cameras in turn.

To provide flexibility, we allow the operator to change from one control policy to another dynamically at any point**.**

3.4 Group- and QoS-based connection manager

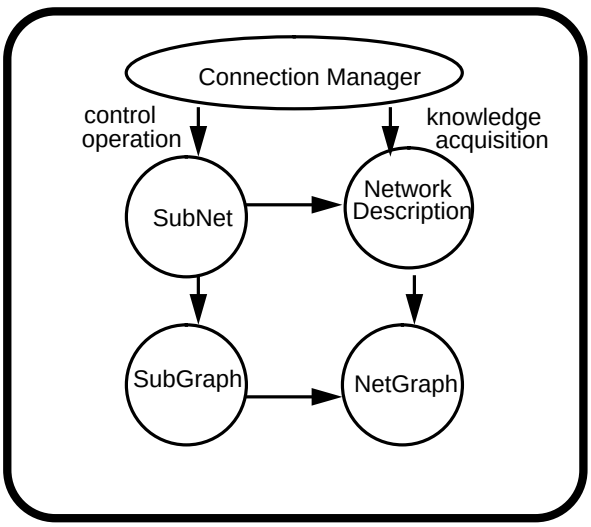
The *connection manager* is responsible for establishing and tearing down connections. For applications such as the surveillance camera and video conferencing systems, it is desirable to use group-based connection control policy. In this policy, when establishing connections for multiple sources or sinks, maximum number of shared virtual channels are seeked to reduce resource usage and network traffic. Such a connection group is called as a *connection closure*. This policy also enables rapid switching between different sources or sinks since only part of the connection needs to be torn down and the rest is reused when establishing a new connection. In addition, the connection manager should be able to find connections that best match users' requirements on quality of service (QoS). In order to achieve this, the connection manager must have knowledge of the topology of the virtual network and the QoS characteristics of all individual virtual channels. It acquires this knowledge when the virtual network is being established/ updated. This section describes the implementation of such a connection manager in the surveillance camera demonstrator.

In the demonstrator, there are multiple sources (cameras) and one single sink (monitor). At any time the monitor can only display input from one camera. The connection manager provides the following functionality:

- Select a subnet for given cameras with QoS requirements
- Reset the subnet
- Establish a connection closure with refined QoS requirements
- Reset the connection closure
- Establish a route from a given source to the sink
- Switch from one source to another

As illustrated in figure 3.4, the connection manager is implemented by four objects: *NetworkDescription, SubNet, NetGraph* and *SubGraph*.

Figure 3.4: Structure of the connection manager



A virtual network is represented internally as an adjacent matrix of a directed graph. This is stored and maintained by *NetGraph*. *NetworkDescription* maps string representation of a virtual network (switches and channels) into its internal representation (vertices and edges) and provides operations to add and remove switches and channels. QoS characteristics are mapped as weights of edges. *NetworkDescription* is invoked during the process of establishing/updating a virtual network to acquire knowledge of topology and QoS characteristics.

SubGraph is responsible for computation of subnet, connection closure and route. Similar to *NetworkDescription*, *SubNet* is responsible for the mapping between the internal representation to string representation of subnet, connection closure and route. It provides an interface to select/reset subnet, establish/reset connection closure, select/switch a source (camera).

In the current implementation, bandwidth and cost are used as QoS characteristics. It is assumed that a virtual channel with lower bandwidth costs less. Users specify their QoS requirements in terms of minimum bandwidth requirements where “high” is mapped as not less than 700 kbps, “normal” as not less than 500 kbps and “low” as under 500 kbps.

In connection control operations, a variant of *all-lengths* matrix algorithm is used to assist finding out whether there exists a through route from a source to a switch and the number of channels and the cost between them.

Subnet selection is realised by removing the virtual channels that do not meet the QoS requirements and the virtual switches that have no through route from the specified sources.

The general idea of source routing is used in connection closure establishment. Since there is only one sink in the demonstrator system, it is simpler to use sink-driven routing. A variant of *weighted spanning tree* algorithm (as illustrated in figure 3.5) is used to find a connection closure between the sink and a group of given sources, with maximum number of shared channels, lowest possible overall cost and satisfied QoS characteristics.

Figure 3.5: Algorithm to find a connection closure.

```
public class SubGraph {
    private static NetGraph g: // graph to process
    private static NetGraph sg; // result graph

    private static void
    getClosureBody(Vertex v, Vertex src[])
    {
      if (v is one of src) return;
      find vertices vs which has through routes from maximum number
          of elements in src among all vertices immediately connected to v;
      choose a vertex vn among vs with lowest overall cost for all through
          routes from elements in src;
      add edge (v, vn) into sg;

      if (vn has routes from all elements in src)
          getClosureBody(vn, src);
      else {
          newsrc1 = elements in src which have no routes to vn;
          newsrc2 = elements in src which have routes to vn;
          getClosureBody(v, newsrc1);
          getClosureBody(vn, newsrc2);
      }
    }

    public static NetGraph
    getConnectionClosure(NetGraph gr, Vertex sink, Vertex src[],
                         QoS req)
    {
      g = gr;
      remove all edges which do not meet the QoS requirements from g;
      compute all-lengths matrix for g;
      getClosureBody(sink, src);
      if (sg does not include all elements in src) sg = null;
      return sg;
    }
}
```

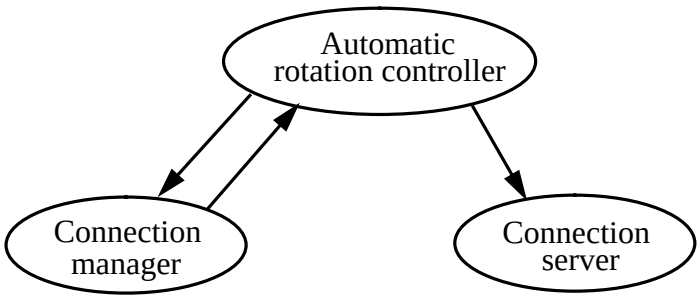
The procedure of selecting a source is to find a route between the specified source and the sink from the connection closure and set up the connection. The procedure of switching to a new source is to find out the virtual channels that need to be torn down and those that need to be set up from the connection closure and act accordingly.

3.5 Automatic rotation controller

As a default control architecture, the demonstrator system provides automatic rotation control.

The controller provides operations to start/stop rotation and reset the server. To start rotation, the controller is invoked with arguments about sink, sources and time slices for different sources. The controller then invokes the connection manager to select/switch sources and invokes the connection servers to set up/off connections. Figure 3.6 shows its interaction with other servers.

Figure 3.6: Interaction between the automatic rotation controller and other servers



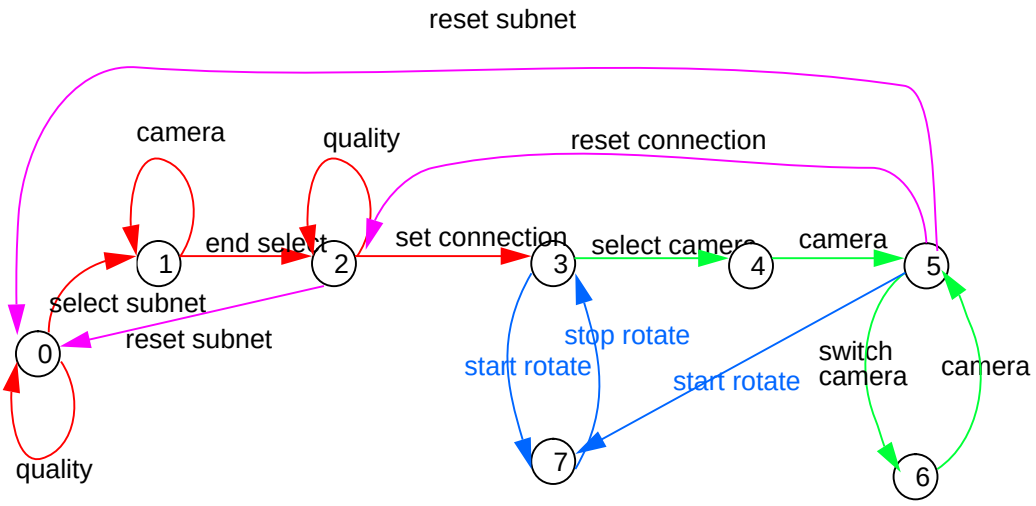
3.6 Operation procedure

The operation procedure can be divided into three phases:

- Select a subnetwork that includes the interested cameras from the whole network.
- Set connections from the monitor to the selected cameras.
- Display on the monitor the video from selected cameras one at a time. The camera being displayed is selected either manually by the operator, or automatically by the rotation controller.

The procedure and operations are illustrated in figure 3.7.

Figure 3.7: State machine of the operations



In the subnetwork selection phase, the operator chooses the cameras he intends to watch. He can also specify some constraints that the selected subnetwork must satisfy, for example, the minimum requirements for the quality of the video. The connection manager will select a subnetwork that includes all the connections, which meet the minimum requirements, from the monitor to the selected cameras. The purpose of this phase is to reduce the size of the subnetwork, and hence simplifying and speeding up the work of the next phase.

In the connections setting phase, the operator specifies the required quality of the video. Then the connection manager chooses, from the selected subnetwork, a group of best connections from the monitor to the given cameras. The meaning of “best” is explained in Section 3.4.

In the displaying phase, the operator chooses the camera to be displayed by using the select camera and switch camera operations, if the manual control architecture is used. When the automatic rotation control architecture is used, the system will automatically switch from one camera to another in a round robin fashion, until the stop rotation operation is called. The operator can change from one control architecture to another any time in this phase. Changing from manual control to automatic rotation can be done by simply calling the start rotation operation and changing from rotation mode to manual mode can be done by calling the stop rotation operation.

4 Summary

In this report, we have presented a simulation of a surveillance camera system to demonstrate the benefits and feasibility of a flexible network architecture. Although it is impossible for us to get any performance statistics due to its nature of simulation, the flexibility of the architecture has been shown clearly. Particularly, the following new concepts for flexible network architecture have been proved through the simulation: switchlet, group-based connection control, QoS-based routing, and multilevel architecture.

From the demonstration we also identified some important issues that must be resolved in order to put the above technologies in practice.

- The first issue is dynamic code uploading. In order to enable applications to dynamically deploy a control architecture to a network, we need to make it possible to upload code to the switch dynamically. Although JavaSoft's *servelet*s allow users to upload code into a Web server, it is not applicable to general application servers such as a switch.
- Moreover, there are some security issues to be solved. For example, how to ensure that only authorised users can upload code to a switch.
- Another issue is the resource reservation and management. To implement the switchlet concept, we must find a way to reserve a subset of the physical switch resources to a switchlet and to ensure that a switchlet can only access the resources reserved for it. This may require some special support from underlying operating systems. For example, the Nemesis operating system [Edwards96] provides this kind of capability.

In summary, although there are still some issues to be solved, we have shown a flexible network architecture that can meet the challenge from emerging applications. A real demonstration based on an ATM network would allow us to collect some performance statistics and investigate more practical issues.

References

[Crosby95]

S. Crosby et.al.,*DCAN report on workpackage 4 switch control protocol*, University of Cambridge Computer Laboratory, December 1995.

[Edwards96]

C. Edwards and D. Hutchison, Approaches to connection management within broadband networks, In Proceedings of third communication networks symposium, July 1996.

[Hanssen97]

O. Hanssen, *FlexiNet - QoS Investigation*, Technical report APM.1977, June 1997.

{Herbert94]

A. Herbert et. al. Scalable Distributed Control of ATM Networks, Project proposal, University of Cambrdge, Computer Laboratory, UK, 1994

[Merwe97]

J.E. van der Merwe and I.M. Leslie, *Switchlets and Dynamic Virtual ATM Networks*, In Proceedings of the fifth IFIP/IEEE international symposium on integrated network management, May 1997.

[Reed97]

D. Reed and R. Fairbairns, *Nemesis the kernel overview*, University of Cambridge Computer Laboratory, UK, 1997.

[Rooney97]

S. Rooney, *The hollowman an innovative ATM control architecture*, In Proceedings of the fifth IFIP/IEEE international symposium on integrated network management, May 1997.

[Tennenhouse97]

D.L. Tennenhouse, et. al., *A survey of active network research*, IEEE Communications, January 1997.