Proceedings of the DARPA Active Networks Conference and Exposition 2002, DANCE 2002
San Francisco, CA, May 2002, pp.409-426, IEEE Computer Society Press (unformatted version)

A Framework for Building Complex Netcentric Systems on Active Network

Javed I. Khan and Seung S. Yang

Media Communications and Networking Research Laboratory
Department of Math & Computer Science, Kent State University
233 MSB, Kent, OH 44242
javed@kent.edu

Abstract

Active applications can add value to communication. Yet it involves sophisticated domain knowledge and complex development process. In this paper we discuss a framework for the construction of complex active network applications. Its design motivation is to enable an expert to develop complex active applications as a value added communication service and make it available for repeated use by the end-applications. As a test case we select a self-organizing adaptive video transcoding channel. This multi-component multi-tier novel active application enables video to propagate over extreme network with highly asymmetric link and node capacity. At the same time it offers a unified channel abstraction to its service subscriber. In this paper we focus how this channel abstraction can be composed within the proposed active service composition framework.

Key Words: Adaptive Video, Netcentric Applications, Active Network.

1. Introduction

In conventional packet network a network junction node (such as a switch or router) only forwards (and occasionally drops) packets [2,9,10]. **Active Network** extends their classical role to a new dimension. Here nodes are also capable of transforming packets in transit through active processing. Active network paradigm can potentially spark a new generation of smart networked applications—which otherwise are not easily realizable on traditional networks.

Network embedded logic can provide unique advantage in communication. For example, a system can adapt with respect to local network states (such as bandwidth or congestion) inside network. Embedded components can also provide high-level localization services (such as language, weather localization, etc). It can also improve the performance of traditional communication service with novel application level knowledge and techniques. For example an active prefetch proxy can dramatically accelerate web surfing. An embedded cache/mirror can help in creating the abstraction of normal communication over time-lapse links. It can add scalability to group communication (by performing application knowledge enhanced data merging and filtering). None is easily attainable from end-to-end paradigm.

In last two years, we have developed and experimented with a number of active applications. Though in this paper we present the case with the transcoder channel, but this framework is the result of our experience with a number of complex active applications (active prefetch-proxy [17], daisy-chain forwarder [14], harness group communicationware [16], etc). To succeed, the active network paradigm must provide an overall application development framework that can deliver software engineering advantages to active applications with network embedded components.

At the lowest level of computing engine an active network requires OS extension where a programmable module is run on router hardware. However, building cost effective and efficient active application on top of it uniquely calls for

research in multi-party development formalism with complex specification interfacing mechanisms and unique system level requirements. Indeed many of the unresolved issues in active networking concerning security, fault-tolerance, and performance cannot be addressed without the overall consideration of the active application and service development framework.

1.1 Background and Related Work

It will be interesting to look into the current research in network centric applications and systems area. Any network-based system is a composition of two types of elementary constructs-- the *process* and the *channel*. A number of research areas-- including parallel processing and middleware, have explored system building formalisms with distributed processing components (from PVM, to DCE, RPC [18,19], DCOM/COM, CORBA [20,21], Java/RMI) [22,23].

More recently, active networking is exploring the means for adding programmability into network data path. Traditional network elements that have been deployed until now perform only a limited set of basic operations on the packets (such as forwarding and routing, fragmentation, packet dropping). Among the techniques for network programmability, Active Network proposes the generalization of the concept by incorporating almost unlimited programmable packet processing ability into the network elements. Here not only the header elements but also the contents of transiting packets may be revised, augmented, compressed, combined, or otherwise processed in transit. At least eight systems have been explored. Examples include Smart Packet [31], ANTS [10], PLAN [32], NetScript [30], and our Virtual Switch Machine [6,14]. The first generation systems have addressed the issues such as execution environment of a foreign code, remote installation and capsule deployment, programming language, router level intercept of active packets [26,27,28]. Pioneering active network platform research now provides valuable insight into the base requirement-- executability of code on routers.

In this paper we discuss a framework called **Active Channel Framework** (ACF) formalism, which will address some of the issues of complex system development. A number of issues from the overall system development framework have remained relatively untouched. These include reusable communicating component based development, dynamic pathway planning, adaptive deployment, dynamic module relocation, and interfacing with service subscriber application and network. Whereas active network presents a powerful but bare-bone architecture to insert and execute programs at junction points, the Active Channel Framework provides the mechanism to build reusable service on it.

In this paper we will explain the proposed formalism by example construction of a novel system – the nomadic self-organizing transcoding channel. The nomadic self-organizing transcoder represents a new generation of network aware reactive system. This is a full logic MPEG-2 ISO/IEC 13818-2 video stream rate adaptation system developed at Kent Medianet Lab [4,5]. As a reactive system, it requires different network service model for automatic launching, management, and seamless operation, which is not present to-date. The proposed active channel framework provides the development formalism for this concept system with support for advanced composition features.

The next section begins with the discussion of the specific challenges of complex systems engineering. We then also briefly explain the features of the complex adaptive features of the transcoding channel. In Section 3 and 4 then we explain the ACF formalism and how this specific system can be constructed and operated within ACF.

2. Framework Issues in Complex Netcentric Systems

Systems Engineering: If we look into the development of the current systems research (middle ware technology), we will see most of the previous formalisms for building complex networked systems have eventually focused on building more complex *process construct* and used standard elementary transport channels for connectivity even at the highest level of process abstractions. What is missing is the polymorphic abstraction of the *channel construct*. A software system construction framework imposes a set of *ontological* and *epistemological* constrained on the *components* or *process* and a set of *protocol* and

lexical constraints on their *rules of interaction* [25]. A well-defined language for constraint specification also provides a means to express properties flexible enough to be felt as programmable yet disciplined enough to ensure **inheritance** and **polymorphism.** Most of the recent research initiatives in middleware technology build on the impressive advances in component abstraction of Object Technology [18-23]. Ironically all approaches focus on providing ingenious domain specific variants-- but still non-polymorphic instances of channel constructs. While, conceptually active network provides a platform for building complex communicating systems yet a polymorphic framework with inheritance property which includes both process and interaction construct is missing.

Planning: In conventional channels the network embedded components are standard based and preinstalled. On the other hand almost by definition in active service composition the components have to be deployed dynamically in the pathway between the source(s) and sink(s). Consequently, a unique stage in active application construction is pathway planning. Besides the actual mechanism for installation, this would require topology discovery, node selection process, node and pathway query. A key challenge here is the provisioning of a mechanism for application knowledge induced and yet deployment time mapping. The planning specifications are expected to come from dynamic network states at the time of instantiation. On the other hand, the constraints are to be specified by the service designer at design time. Consequently, a key requirement here is the appropriate specification language. The planning process itself may require additional system level network embedded meta-components. These meta-components however, may not be all generic as they too can require application specific scouting.

Network State Accessibility: A service with network embedded components is meaningful if the embedded components can dynamically react with respect to network local states. This imposes several new requirements, which are not so acute in classical end-to-end paradigm. There is a historical irony in the design of network software stack. In classical design each layer in the network software stack has been strongly isolated from its upper layer. For simplicity the interlayer interfacing mechanisms were designed as interaction free service transaction pathway. This model simplified development of first generation applications. The flip side of this design choice is that now therefore network states are very difficult to access. For building efficient active applications system level re-provisioning will be required so that embedded service components can be mutually secured and trusted, and yet will allow open standard-based access to network local states. A generalization of the requirement also calls for an inter services state-interfacing. Here a trusted component from one service should be able to pass on vital local states to a collocated component of another service.

Dynamic Service Configuration: As a part of a large scale dynamically deployable network service, it will be more prone to dynamic component relocation. The issue of run time service recomposition is not a simple extension of the first time service installation. In service reconfiguration additional constraints are placed by the requirement of minimum interruption of the ongoing service sessions. For example, while intermediate components are relocated it must be ensured that an ongoing service does not suffer from loss or duplication of packets, or loss of duplication of processing cycle etc. We must reduce the impact on overall processing and communication delay. The signaling sequence to install/ uninstall components within a running service requires application specific sequencing and deadlock avoidance.

Separation of Service Development and Service Subscription Interface: Another critical component that arises in active service composition is the creation of a powerful middle-layer abstraction. This is related to the remarkation of the conventional boundary between end-to-end applications and network. While, the active service itself can be a complex system, however, the subscription and use of the composed service (once designed) should be easy and intuitive. This required provisioning on a middle-layer abstraction, where third-party developers can develop the components required for active service, while the end-point users and applications can take advantage of these services with an easy to use interface. This will require two distinct programming interface design- one for the service designer another for the service subscriber.

With these requirements in mind we have proposed the Active Channel Framework (ACF) for developing active applications. In this paper, we illustrate the ACF formalism with a concept application that concentrates on creative adaptation.

3. Nomadic Video Transcoding

3.1 Adaptive Transcoding

As a test application, we present the nomadic MPEG-2 [4] rate transcoding mechanism. The application system has a number of complexities to offer substantial challenge to the framework.

First of all it addresses the issue of adaptation from two levels. It adapts with respect to the local constraints of two critical network resources—bandwidth and the processing resource at the junction nodes. As opposed to classical server-client model for building network applications under current network software architecture, this application has three parts-- server, transcoder and the client. The middle component is the channel and is reusable. It can be launched as a set of active capsules in a suitable active junction point in the stream pathway and downscale it if needed. The resulting system has two adaptive features. It performs an adaptation function for its subscriber. Secondly, its can evolve its own architecture based on the network resource.

3.1.1 Functional Adaptation

The transcoder senses local asymmetry in link capacities at various junction points of a network. Accordingly it auto-converts the video stream rate. For example, it can be dynamically deployed at nodes splicing a fiber and a wireless network, and thus it enables an incoming high-bandwidth video multicast stream to be re-encoded for the outgoing low-capacity wireless links.

This particular type of adaptation demonstrates advantage of active application paradigm and does not have easy solution in end-to-end-paradigm. In comparison, today's media servers are fixed rate based. They store multiple copies of the same media one for each supported rate class (LAN, DSL/Cable, 56K, 28.8K etc.) [13,1,3,4]. The end-user is required to specify its rate. Although there have been few attempts to automate such specification, but the difficulty seems to lie fundamentally with the end-to-end network programming paradigm. Neither in a live video multicast, nor in stored video cache, the end-to-end solutions seems satisfactory. Classical solutions such as, sending the high-speed stream cuts off the low speed clients. Sending the lowest speed stream penalizes others by forcing the lowest quality to all. Sending multiple streams, burdens the network. In contrast the in-stream rate adaptation by a **nomadic active transcoder** can solve this riddle and offer optimality even at the level of individual links.

3.1.2 Architectural Adaptation

The system also has been designed to adapt with respect to node computation power. This is also a unique problem specific to active applications. Video stream rate adaptation is a challenging task. Because of complex inter packet data dependency, packet level rate adaptation—which is almost blind, offers very limited **down-scalability**. Dropping only 20-25% of the UDP packets can render an entire stream useless [12,2,24]. Content unaware, just packet slicing backed rate control seems to very wasteful and limited in their efficacy. Smarter content aware transcoding, in contrast can offer much broader range of rate adaptation with much graceful loss of video quality.

However, MPEG-2 transcoding is also quite complex and computation intensive task. A number of techniques have been investigated for accelerated transcoding by us and other researchers, such as motion vector reuse, fast DCT domain transcoding, parameter bussing [11,7,8,24] These models provide a three-way tradeoff between the computational complexity, video quality degradation and second stage compression. It seems that in near future with the rapid advancement of the VLSI technology some nodes may be able to garner enough processing power for real-time high fidelity video transcoding. However, on any given large scale network the fact of the matter is that there will be always inequality of processing capability just like the asymmetry in bandwidths. Consequently, this nomadic transcoder also demonstrates **self-organization** behavior and choose the right operating state in this three-way trade-off based on the available

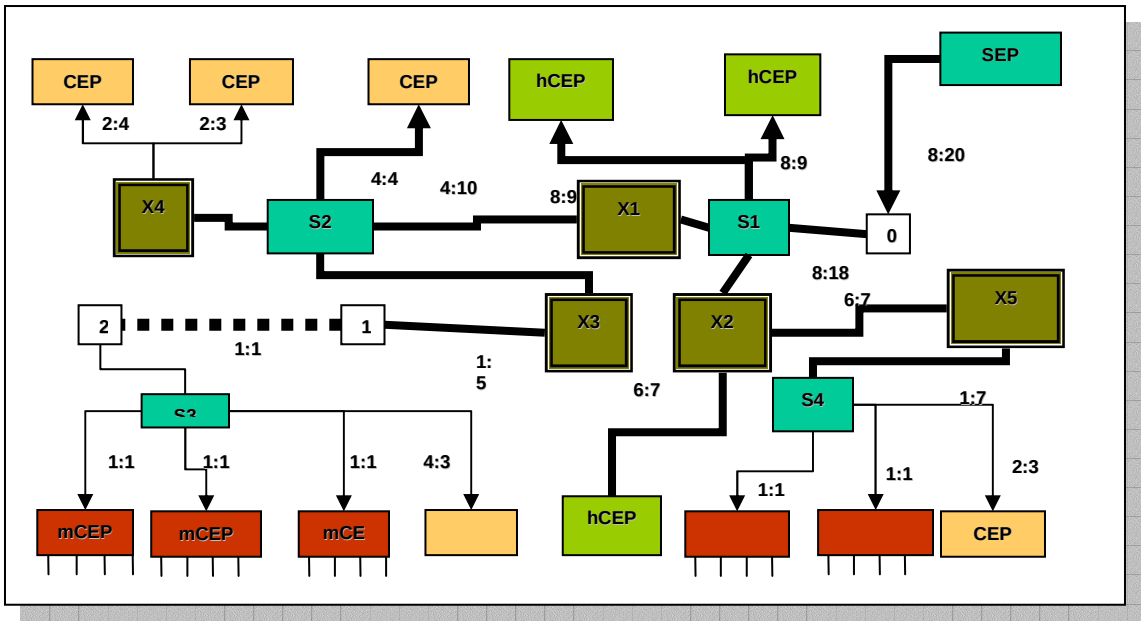

Fig-1 An adaptive video distribution scenario in a network with asymmetric capacities. There are various types of clients. High-definition client (hCEP) have high requirements, and mobile clients (mCEP) delivery. These are also connected with links of different capacities. An distribution network with embedded rate transcoders (X) and stream splitters (S) at correct network points can facilitate the distribution.

network resource of processing power at the junction points[1].

3.1.3 Service Interface

Also, the overall system creates a service abstraction of an MPEG-2 video communication channel where instream rate adaptation is automatically available between a video source and sink groups, and is reusable by any high-level video application, without interfering with its functionality or without reengineering of the transcoding system's code base. Also, the system components should automatically be deployable in the correct nodes as per service designer's specification in the network without active intervention from the user application.

3.2 Nomadic Transcoder

3.2.1 Components

The model we are investigating has three service modules (a) application server (b) transcoder channel and (d) the client. The transcoding channels has two types of functional components—a splitter and a transcoder. A splitter duplicates a stream in a junction. A transcoder downscales incoming video rate.

Fig-1 shows a sample video distribution network with asymmetric link capacities and a typical deployment of the modules that builds a distribution tree. The server is attached to the *server end-point* (SEP). There are various client classes (regular, mobile and high definition), which are attached to the network via *client-end points* (CEP), *mobile client-end-points* (m-CEP), and *high definition client-end-points* (h-CEP). Each end-point can in turn act as a local distribution/ concentration center. The network connecting these end-points has unequal link and node capacities[2]. Each link is labeled using two quantities R:B where and B is the link bandwidth,

[1] Node capacity adaptation has been little addressed until now, though it is particularly important for active networking.

[2] Bandwidth is increasing with the rapid advancement of technology [29]. However the effect seems to be limited at the core. Effectively what is increasing dramatically is the heterogeneity and asymmetry in the Internet. The augmentation of non-traditional devices [33] in the Internet will further intensify the situation.

R is the requested video rate at the leaf links or final flow rate at the internal links.

3.2.2 Module Mapping

The first application issue is the optimized module placement. The video rate in the links should be calculated bottom up based on the requests from the CEPs downstream and the available link bandwidths. At each junction point, the downstream requests are aggregated and the maximum of these are propagated upstream as a new request. The requested rate either increases or stays the same as it propagates upstream. The number of required transcoders is equal to the number of rate steps inside the distribution network. However, instead of catering to each requested rate, the number of transcoders can be minimized by using rate classes. A possible placement of the transcoders seems to be at the junction nodes. This requires each m-way junction node to have m-1 step down transcoders. However, each step down requires significant computation. Therefore, to distribute the computational load the step-down transcoders are not placed at the junctions, but one node down along each of the downstream paths.

3.2.3 Dynamic State Information Exchange

The placement optimization depends on local network states. These include link bandwidth, congestion, switching speed, processor speed, and available computational power of the nodes. The link capacity allocated to the video distribution can also vary with time such as due to congestion. Similarly, the computation cycles available to the adaptation mechanism such as the transcoder can also vary based on the multiprocessing load at the nodes. The transcoders must be able to optimize its rate conversion factor in response to the change in link and node states up and downstream. Similarly, the transcoding operation can be performed at several quality/computational effort choice levels. The particular level should be chosen based on the available computation power at the node. Thus a prerequisite to any adaptive system is dynamic and efficient monitoring of various local network state information.[3].

[3] We have developed a group state communication ware (another active service) [16] and this ware can be invoked by the transcoder channel manager if needed. There are other management tools such as NESTOR, which can be potentially used [34].

3.2.4 Dynamic Module Relocation

Initial module mapping is not final. Any change in the local state not only can spark change in step-down rate, but it can effectively require activation of new and deactivation of existing transcoder modules. Dynamic departure of clients can similarly trigger activation/deactivation of splitters and transcoders. Congestion, reduction in CPU cycle allocation may necessitate migration of transcoding operations to neighboring nodes. Thus, ideally, in a large adaptive system these modules should be occasionally able to relocate, however, without apparent disruption of the basic stream. Relocation should not cause loss or duplication of data. The information order should also be preserved. Additionally it is desirable that it should not impose significant delay. System stability should also be maintained avoiding excessive reorganization.

4. ACF System Architecture

4.1 Overview

While the above explains the target that we would like to achieve, however in a large network our principal interest is not a single smart application but a formalism that can automate the entire placement, deployment and running process of the above system. Consequently, we first identify the critical network layer services needed for adaptive systems. We then present the proposed organization for the entire system.

First we envision the rate adaptive service to be a special and generalized form of abstract communication. While, traditional networking provides only two types of communication channel constructs TCP and UDP, we generalize the concept and consider that the automatic rate adaptive communication service indeed is a higher level of communication service to the applications. The channel construct provides two important benefits. First it provides a separation between the network related and the network independent tasks. Secondly, it allows the entire design and modules to be reused by various actual server and player

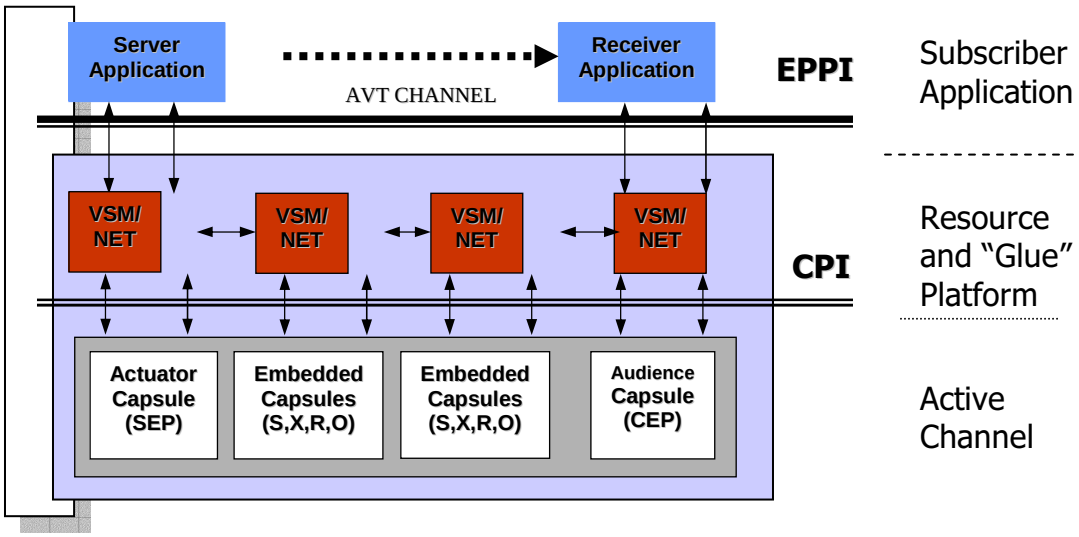

Fig-2 The three-tiers of system architecture in ACF formalism. The top is the subscriber application. It extracts the communication service from the underlying active network via a set of embedded capsules.

applications. For example, while, in actual server application VCR operations (such as fast forward, rewind, pause, etc) can be supported, the network layer modules are only concerned about the transport aspect of the service. We will call it rate *adaptive transcoding video* (ATV) channel.

Based on this channel abstraction we propose a three-tier system architecture. Fig-2 illustrates the tiers.

First is the subscriber application itself. More specifically, the network independent part of the domain routines resides in this tier. The application advanced auto-rate adaptive transport service connecting the end-points, however, it does not want to be concerned about the implementation details.

Second is the channel layer. This is the layer that houses the reactive components that bridge between the network and the application (shown in the bottom box). The modules in this layer are programmable however; they execute strictly under the control of the network and help in the local state dependent adaptation.

The third is the enhanced network layer, which acts as a "glue" layer connecting the application layer, channel layer and the actual network and performs automation and facilitation. It is placed on top of the traditional network and the OS in the individual machines. We will refer to this glue layer as the *network operating system* NOS. This indeed has three parties. Besides network, this has a converted active router called Virtual Switch Machine [6] and the active system building and execution environment (BEE). The BEE provides the added extensions and utilities to realize the development formalism. We denote this combined system as the *network operating system* NOS (NET/VSM/BEE).

The *adaptive transcoding video* (ATV) channel is built with four primary types of capsules (a) actuator capsule (server-end-point SEP) (b) splitter (S) (c) transcoder (X), and (d) audience capsule (client-end-point(s) CEP).

The actuator and the audience capsules interact with the applications, and at the same time co-ordinate with other capsules and hide internal details and provide a single channel abstraction to the application. The CEP modules can have multiple versions specializing to attend mobile or high-definition clients adapting to their specific needs. There are also two maps, which describe the rules for placement of the primary capsules and their interconnection, and two auxiliary capsules, which dynamically evaluate the network dependent placement logics and decide the roving behavior and the self-organizing states of the nomadic components. The channel designer thus can be different from the application programmer.

A multicast video application requests an adaptive transcoding video (ATV) channel in a similar way it requests sockets. NOS installs the capsules and creates the requested channel and application receives the finished product.

4.2 Task Division & Information Fragmentation

The tri-partite development model helps in the task separation in the development process and is the key to any possibility of complex application engineering. However, the framework design becomes challenging because of the knowledge fragmentation in multiparty development. Let us first look into the fragmentation. Then we will explain how the system glues them together. We will call each a sub-domain. Below we describe each sub-domain's information structure.

4.2.1 Channel Designer

The channel designer generally knows very well about the operational mode, control flow and detail architecture of the capsules. S/he also knows the situational constraints of the channel components. However, the channel designer does not have access to the exact network map on which the system will run. S/he knows only in a logical way on which type of topology the system can be deployed. Channel designer also has knowledge about the computational and communication power that will be needed to support the channel service. S/he also knows the nature of service it can provide. S/he probably has a metric defined to quantify the service it can provide. However, since it does not know the exact traffic that a specific instance will carry, or the exact topology on which an instance of the channel will run, therefore it can only provide the deliverable service capacity, and the required network resources to support it in some canonical way.

4.2.2 Subscriber Application

The subscriber application on the other hand is eager to remain completely ignorant about the channel architecture, capsules, their connections, nor does it is interested to know about these details. It is mostly interested in the service the channel provides. The subscriber on the other hand is the first party to know the locations of the end-points. However, it does not know, nor is it interested to know the underlying network topology. Also, the subscriber application is the most knowledgeable among the three about the characteristics of traffic that will flow through the channel. The subscriber is expected to have a fair idea about the nature of service it will receive from the channel, though it may not know the exact metric that the channel designer has used to quantify it. It is interested to know the metric, so that, if it wishes it can specify its requirements. It also reserves its right to know how much it is actually getting.

4.2.3 Active Node OS NET/VSM/BEE

On the other hand, the active node OS (NET/VSM/BEE) has no prior knowledge about the channel architecture, its components, capacity, or network capacity requirements. It also does not know in advance about the traffic characteristics or size. However, this is most knowledgeable among the three about the underlying network infrastructure—i.e., the actual topology, and the quality and capacities of its various elements. However, it does not know which among them will

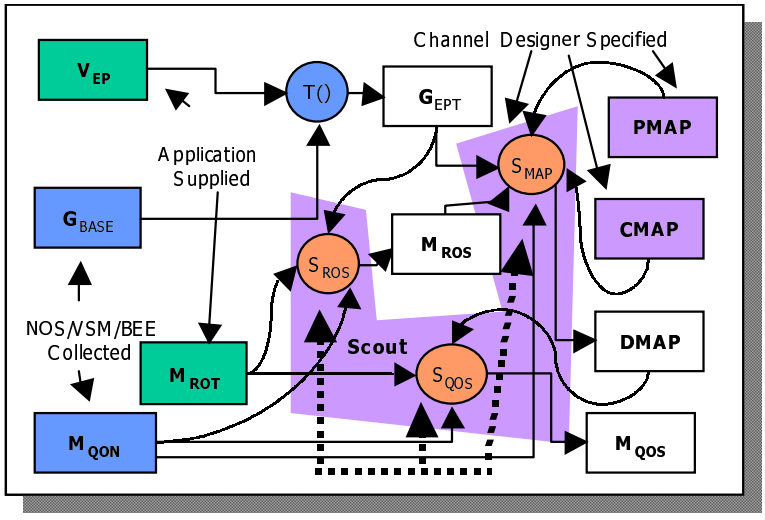

Fig-3 The tripartite information dependency between the application program, channel, and the NOS/VSM/BEE. The logic for S() functions are supplied by the Channel Designer as a special capsule set called Scout.

be the end-points. Nor does it know in advance which part of the network will be used by the channel components and what might be the domain specific criterion for selecting the sites.

4.3 Elements and Dependency

It becomes the responsibility of the Channel Building and Execution Environment (BEE) to glue together pieces of these fragments so that individual parties do not need to know other's components, while at the same time, each party can receive the required information needed to perform its designated tasks. Fig-3 explains the dependency and the scheme.

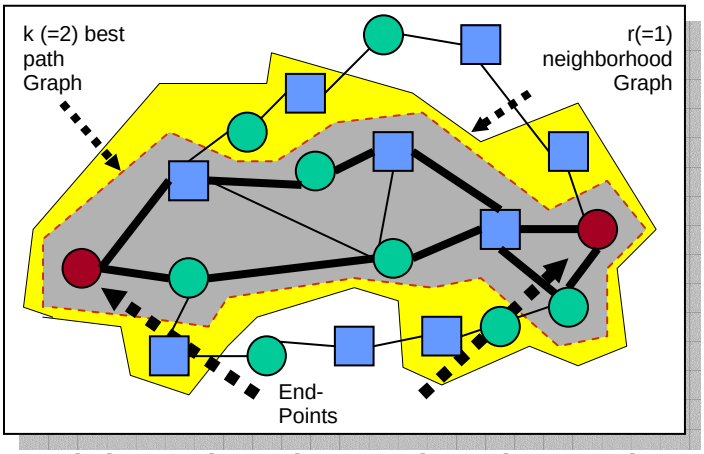

Fig-4 the base topology and corresponding End-Point topology graph.

The Subscriber Application Supplied elements are End-Point Vector (V_{EPT}) and Quality of Traffic Metric (M_{ROT}). The End-point Vector specifies the location, and channel specific type of the End-Points which have to be connected in the channel. M_{ROT} is an optional parameter by which the Subscriber Application may express the requirements of the traffic (using the quality-of-service metric of the channel service).

The underlying NOS (NET/VSM/BEE) is responsible for supplying the Base Topology Graph (G_{BASE}), and the Quality-of-Network Metric (M_{QON}). The base topology graph is the network connectivity information where route exists. Nodes not included in the base topology cannot be connected. Network resources not included in it will not be considered for inclusion in the channel. Fig-4 shows an example of base topology between two end-points.

The QON metric provides a snapshot of network resources in peak and current values. For link resources we track bandwidth and delay and for node resources we track processing power. Typically NOS should use a separate system utility to collect the metric.

The channel designer provides the capsules required to establish the service. It also provides two canonical maps called PMAP, and CMAP specifying the initial placement and connections of the static capsules. The channel designer also provides a capsule called "Scout". These capsules serve as the intelligence gathering tool for the channel designer in his/her absence. It contains the channel specific logic to determine the location and activation states of the dynamic capsules.

The above configuration is sufficient to determine the other required system information. Fig-3 shows the information elements and their dependency in ACF. Once the end-points are declared the Installer component computes the End-Point Topology graph (G_{EPT}), which is a pruned sub-graph of G_{BASE}. It contains k-best path between the end-

Placement Map

```
DEFAULT URL       =shttp://medianet.kent.edu/capsules/*.cap
MAP VALID ON      =tree, path.

Rover     LOC=all nodes       INVOKE= 200s.
Organizer           LOC= all nodes with rover.XCODE=ON INVOKE=5
SEP       LOC=src
          URL=shttp://tv.kent.edu/capsules/version2/SEP.cap
Splitter  LOC=all junctions between src and sinks
Xcoder    LOC=all nodes with rover.XCODE=ON
CEP       LOC=all sinks

NETORDER          (Rover,Organizer,SEP,Splitter,Xcoder,CEP)
EXEORDER          (Rover,Organizer (SEP|Splitter|Xcoder|CEP))
```

Fig-5 (a) Capsule Definitions and Constraints

points and its neighborhood with radius r. A topology discovery algorithm is used to determine $G_{EPT}=T(G_{BASE}, M_{QON}, k, r)$. The example of Fig-4 illustrates the base graphs. It has both active (square) and passive (circle) nodes. With the declaration of end-points it shows the base topology graph with *k=2*, and *r=1*.

Function $S_R()$ in the Scout provides the Requirement-of-Service metric (M_{ROS}). It tells how much network resources will be required to satisfy the given Requirements-of-Traffic (M_{ROT}) posted by the Application. Function $S_M()$ then provides the dynamic capsule placement mapping (DMAP). Function $S_Q()$ then provides the feedback on the Quality-of Service metric supportable (M_{QOS}) using the current dynamic mapping.

Connection Map

```
Rover.port0→(Request-notify)→ prev.Rover.mport1
Rover.mport2→(Grant-notify)→ anynext.Rover.port3
SEP.port1→(MPEG2-stream)→ next.Xcoder.port0 |
                    next.Splitter.port0| next.CEP.port0
Xcoder.port1→(MPEG2-stream)→ next.Xcoder.port0|
                     next.CEP.port0
Xcoder.port3→(Control)→ prev.Player.port2|
                    prev.Xcoder.port2
Splitter.mport5→(MPEG2-stream)→ anynext.Xcoder.port0|
                    anynext.CEP.port0

Request-notify=     Request:UDP
Grant-notify=       Grant:UDP
MPEG2-stream=       stream:TCP
Control = switch:UDP+RatePar:UDP
```

Fig-5 (b) The Capsule Interconnection Rules

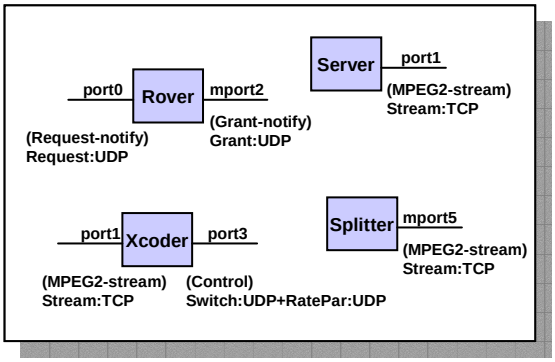

Fig-5(c) The capsule ports.

4.4 Channel Components:

4.4.1 Placement Map:

Placement Map (PMAP) and *Connection Map* (CMAP) together tells the installation manager how to cast the capsules in a given network. The specification itself does not require any knowledge about the specific network topology rather it specifies how the components should be connected on a given class of graph.

Fig-5(a) and (b) shows typical example of these maps. Placement MAP shows rules and constraints for placing the channel capsules. Each of the lines in PMAP describes a capsule object as attribute, value pair. Capsule name, placement constraint, URL for automatic loading, etc. are some of the attributes. Placement constraints can use especially defined network state variables such as *XCODE* (it's use will be explained shortly).

The placement is generally interpreted in the context of a specific *context network graph* (such as chain, binary-tree, multi-way tree, general graph etc.) on which the mapping is valid. PMAP statement *VALID ON* indicates this context. The system assigns a partial numeric order to all the components. A unique ordered id is assigned by the system for each channel session based on the specific network on which the channel is cast. The *NETORDER* statement resolved the partial order among the capsules, which operate on the same node. This partial order is used for topology independent addressing.

The *EXEORDER* statement specifies the invocation order of the modules (if needed) for deadlock free operation, and is interpreted by the scheduler. For example the Rover must be invoked before the others. While, there is no specific invocation order requirement for the SEP, Splitter, Xcoder, or CEP.

Channel designer can also specify an invocation mode for the capsules such as periodic, non-terminating (default), event-driven, etc. For

example the Rover and the Organizer capsules are periodic capsules with invocation period of 200 seconds and 5 seconds respectively.

It the responsibility of the Channel Designer to determine the proper EXEORDER sequence that will enable deadlock free operation for the specific design of the channel. The NOS installer does not guarantee the deadlock avoidance.

4.4.2 Connection Map:

Connection Map (CMAP) describes the rules for connecting the capsules by specifying the connection rules between the port pairs of the capsules. The capsules have numbered *ports* and *multiports (mport)*. All are simplex and uni-directional. A *port* connects to only one other port. A *multiport* is a collection of *memberports* which share the same data. The actual fanout is determined after the casting by the installation process. Each *memberport* of *multiport* set is bound to just one port of another capsule. The sharing modes can be scatter, gather, replicate etc.

Rover Capsule

```
1.  /*Determine the topological location If the Rover is
    co-located with CEP, then  set Player's need is my
    need. It considers the player to be stream 0*/
2.  ….
3.  downstreamrequest[0]=MY_CEP_NEEDS();

4.  /*propagate a request upstream*/
5.  if( not receivernode)
6.  Receive(from all-1st-downstream, in
    &downstreamrequest[l]);
7.  requesting=min(max(all downstreamrequest[l]),
    uplinkcapacity);
8.  Send to parent Send(requesting);

9.  /*Request granting phase begins from the top*/
10. Receive(from 1st-upstream, &granted,&xcoder);
11.       Else granted=requesting;
12.       If(xcoder==ON) rover.xcount++; /*initially zero*/

13. /*and propagates downstream*/
14. For all children I {
15.       If(granted>=requested[l])
16.       rover.granting[l]=requested[l];
17.       Else rover.granting[l]=granted;
18.       If(granted>rover.granting[l]) {
19.                 rover.xcoder[l]=ON;
20.                 rover.xcount++;
21.       }
22. /* Sample controvertial optimization */
23.       If(xcoder[l]==ON & downlink[l]>granting[l])  {
24.                 rover.xcoder[l]=OFF;
25.                 rover.xcount--;
26.                 pushdown[l]=ON;
27.       }
28. ……
29. NOS_SetData(rover);              /* NOS variable
    is updated*/
```

Fig-6 (a) The rover pseudo-code that determines the self placement behavior of the nomadic transcoders

Each entry in the CMAP corresponds to one sending port of the capsules. The capsules for which Fig-5(b) provides the connection constraints are shown in Fig-5(c). For example, it specifies that *port1* of a transcoder (Xcoder) capsule can only connect to the *port0* of the next transcoder module or *port0* of the Player module (SEP.port0 connects to the Player). The relative address 'next' corresponds to the next transcoder unit according to the partial order determined.

The CMAP interpreter evaluates the rules from top to bottom and left to right. CMAP also defines the message structure and connection type. For example it specifies that the "Grant Notify" connection between the Rovers should be implemented using an UDP port. CMAP can define complex connections. For example the "switch" connection is a complex connection. The aggregator in CMAP then defines the packet structures and the elementary transport mode (TCP or UDP) on which the actual connection is established.

4.4.3 Adaptation Capsules:

The dynamic placement and adaptive self-organization behaviors of the transcoder capsules is determined by two 'system' capsules (a) *Rover* and (b) the *Organizer*.

Rovers dynamically determine the nomadic behavior and optimum placement of the capsules inside the channel. As specified in the PMAP execution order (Fig-5(a)), Rovers are invoked before other capsules. To illustrate the process we show a simple Rover body in Fig-6(a). (This is also not the only way it can be implemented). Upon coming to life, Rovers receive the current Topology (G_{EPT}) and their relative locations in it from a NOS call. Each Rover also then receives the link bandwidths of the connect links by using

NOS calls. These Rovers are interested in their uplink bandwidth (variable *uplinkcapacity*). Then all the Rovers enter in the rate request propagation phase. The phase begins from the receiver side and propagates upstream. Rovers co-located with the CEPs are the CEP Rovers. CEP Rovers consider the sinks as their downstream port0 and store the requested bandwidth value in *downstreamrequest0*. The upstream request for all Rovers, including the junction Rovers, is determined by the *min/max rule* min(max(*downstream[i]), uplinkcapacity*)). All Rovers forward this request one step upstream. When a Rover receives a request from i-th children it then stores the requests into *downstream[i]*. All Rovers follows the *min/max rule* for propagating the rate request. Once the request reaches the server-end, SEP Rover grants a specified rate (*granted*). It then propagates downstream. This time each Rover sends the quantity *granting[i]=* min(*granted, downstream [i])* along i-th downstream port. For each Rover where there is a mismatching port with unequal *granted* and *granting[i] Rover.XCODE* is then set to ON.

The Rover in Fig-6(a) also shows a sample optimization. A junction node (which already will host "Splitter") may push down the "Xcoder" one step downstream along the step down port using the last code segment. Before it can push however it checks if there is enough downlink capacity (*downlink[i]).* This generally will save junction nodes from being overloaded. Once, all the Rovers complete, all the nodes those are chosen will have their Rover variable *XCODE* set to *ON* appropriately. In the same process, the Rovers also calculates the step-down ratios for each transcoder which is the *granted:granting[i]*. (not shown in the code fragment). For simplicity we did not show the numerical considerations in the Rover code presented in Fig-6(a). However, we will discuss it in detail later.

The self-organization state of the transcoders is determined by another capsule called *Organizer* (Fig-6(b)). Each capsule instance can save some data for their subsequent instantiations. At the end of each capsule instantiation can save data by NOS_SetData(), and in next instantiation can get back data by NOS_GetData() calls. This saving is node local. This mechanism enables them to keep track of the video frame-rate (*rate*) between two

Self-Organizer Capsule

```
….
/*set the xcoder configuration state value*/
now=NOS_GetData(system.now);
then=NOS_GetData(saved.then);
frame_now=NOS_GetData(xcoder.frameno);
frame_then=NOS_SetData(saved.frameno);
NOS_SaveData(saved.timethen, now);
NOS_SaveData(saved.framethen,frame);
rate=frame_now-frame-then)/(now-then);
SNR=NOS_GetDate(xcoder.snraverage);
organizer.xcodestate=CalculateConfigure(framerate,SNR,
QoS);
NOS_SetData(organizer);/* NOS variable is updated*/
```

Fig-6 (b) The organizer pseudo-code that determines the Self-reorganization states of the transcoders

instantiations. This particular self-organizer capsule checks the current frame number of the collocated Xcoder (*xcoder.frameno*), picture quality statistics (*xcoder.snraverage*) and also the current time (*now*). It then retrieves the frame number and time during its previous instantiation. Based on the outgoing frame-rate from the local transcoder it then modifies the *organizer.xcodestate.* The Xcoders after processing each group-of-picture (GOP) [5] checks for this variable to receive their self-organizing state determined by the self-organizer logic.

4.5 NOS Components

The overall implementation of the system now requires the following services from the NOS.

4.5.1 Programming Interface:

Unlike classical channels, the NOS provides two interfaces. First is the *End Point Programming Interface* (EPPI) to the applications (server and player) to request and run a rate transcoding channel. The other is the *Capsule Programming Interface* (CPI) for capsules to execute and co-ordinate their functions. We have proposed a detail of both the interfaces in [14].

4.5.2 Services:

Below the interface, the following new network layer services are now required for the ATV channels: (a) channel installation service (b) intra-capsule communication service, and (c) network state exchange service.

Channel installation service include the topology discovery, determination of G_{BASE} and G_{EPT}, (Fig-4), the loading of capsules as per CMAP and PMAP (Fig-5(a) and (b)), assignment of execution resources to capsules, and the allocation of partial orders address to capsules. It also needs to ensure secured deployment.

The intra-capsule communication service includes channel identifier allocation, interpretation of channel relative addresses (to be described shortly), inter-capsule message delivery as per the addressing, buffering of saved data between capsule instantiations, provisioning on CMAP placement and other system capsule variables (such as rover.XCODE).

The network state exchange service includes making the node local part of M_{QON} available to the local capsules such as capacity of connected links (*uplink bandwidth* in the example in Fig-6(a)). These also include SNMP link MIB-II variables, and additional active node states such as CPU, memory allocation, and execution time to the capsules. An important point to note is that, the system does not have to make such information available to the capsules on global basis. It is sufficient to supply these to the capsules who are local. From there capsules can take charge and exchange it globally if needed.

4.5.3 Organization:

All services have three NOS components. A manager, which works at the actuator end of the connection, an audience-agent, which works at the audience end-point, and a network agent, which works in the intermediate nodes. Fig-7 shows the service stacks in these three positions. The service stack involved at the junction nodes is little different form the end-point stacks. Junction nodes do not need any EPPI, as there is no subscriber application component there. However, since, the junction points can be a typical active router, instead of relying on general OS, a *Virtual Switch Machine* (VSM) has been created on the router. VSM identifies active communication from regular routing operation and when an active data arrives it diverts the active packets towards the capsules. VSM also allocates the CPU and memory resources between the competing capsules and acts as a capsule scheduler. Immediately above the VSM an execution environment called channel *Building and Execution Environment* (BEE) provides the utilities upon which the capsules actually execute.

4.5.4 Dynamic Relocation:

The channel installation also provides support for dynamic relocation. As a part it provides several additional *passive* and *periodic* capsule instantiation modes. It also provides mechanism for *deferred* capsule cancellation. In the example,

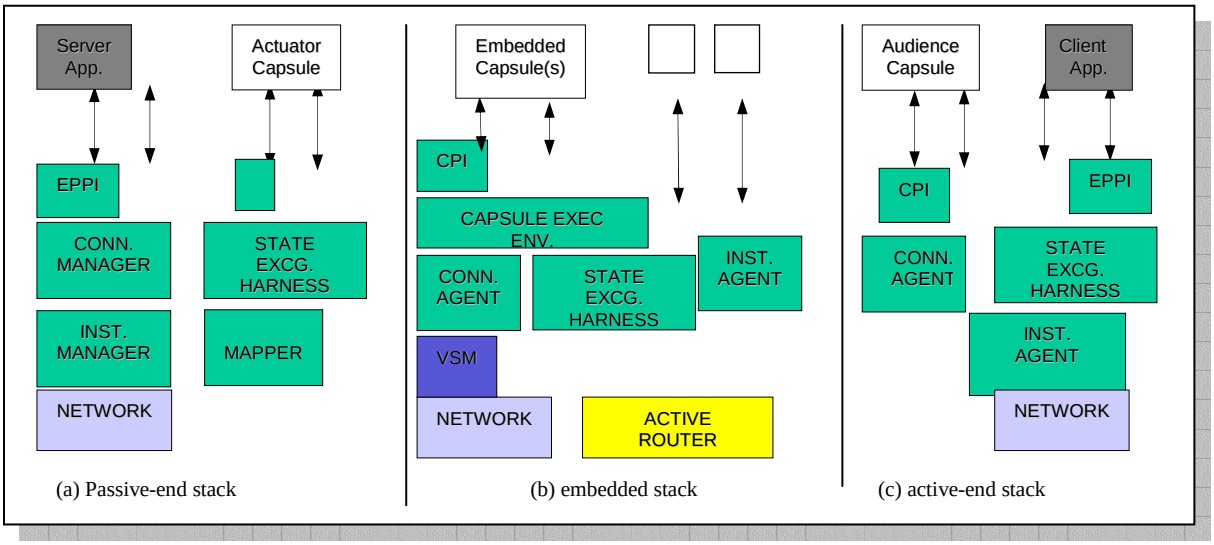

Fig-7 network services for adaptive application maintenance and execution NOS layers

the Rover is periodically invoked by the VSM/BEE. Each time a new calculation is made on the placement variable. If result changes, installer accordingly dynamically starts and stops capsules. For seamless invocation, the new capsule is first loaded and placed in the background without changing the current operation. In passive instantiation mode some capsules can be preloaded but kept in a dormant state,

Table-1 Requirements of Traffic M_{ROT}
Requirements Posted by MediaetServer Session

End point	Required Bandwidth
SEP	30 Mbps
CEP1	30 Mbps
CEP2	20 Mbps
CEP3	10 Mbps

Table- 2 Node M_{QON}
Resource availability on End-Point Topology G_{EPT}(radios = 1)

	A1	A2	A3	A4	A5	A6
MAX	100	200	300	400	100	100
Average	50	100	100	200	100	70
Available	50	100	200	200	300	50

Table-3 S_{ROS} Calculation Table
Network Resource Requirement per unit of Service

Capsules	Needed Computation
Splitter	1 per 1MB video stream
Xcoder	12 per reduce 1 MB

Data marking is used for seamless operation. However, the data flow is marked by applications. For example the AVT transcoder marks the stream in per GOP basis (actually it is already there and in this case no additional effort is need) and ensures that switch occurs after the completion of the current GOP for each module. For the exiting capsules it is stopped by NOS in a deferred basis only at declared cancellation points called *ready-to-stop* point. For incoming capsules there can be similar (optional) *ready-to-start* points. When, the module is ready to operate, only then the VSM switch is invoked to route active packets to the newly activated module. Seamless revocation is performed by disabling the packet forwarding first. For graceful revocation the VSM/EE invokes a particular capsule module called *CAP_Destroy()*, given by the channel designer for graceful departure.

4.5.5 Channel Relative Addressing:

In addition to the above services, the VSM/BEE also provides an abstract addressing scheme, where capsules and maps can use a channel relative addressing mechanism to refer to each other based on their logical position in a channel, without knowing the actual deployment. The addressing scheme has three references-- *src*, *sink*, and *this node*. Complex references are built on them using set operators and modifiers such as *upstream, downstream, all* etc. Once, the actuator and the audience end-points are determined, the CRA interpreter returns determined the actual network addresses for all CRA addresses. The CMAP, PMAP and the Rover shown in Fig-5 and Fig-6 contain several examples of such network relative references.

4.6 Channel Instantiation

Finally, Figure-8 shows an example assignment of the capsule components for a specific channel instance using the ACF.

4.6.1 Request

The instantiation starts with a subscriber application "MediaVideo" which would post the initial end-points (in that case one server-end connected at entry point A1 and three player-ends connected at entry-points A2, A3 and A6) via the EPPI interface. End-points can also join (or leave) dynamically later. The application requests an "AVT channel" and posts its M_{ROT}. A sample *requirements-of-traffic* metric is shown in Table-1.

4.6.2 Building

The Installer determined the End-points Topology G_{EPT} from the base topology G_{BASE}. Table-2 shows a sample M_{QON} metric. It obtains the PMAP, CMAP, security certificates and the capsules for the AVT channel from code server. Next it launches the Rovers. It also launches the "Splitter" on the only junction node A2, and SEP on the MediaVideo Server entry point A1, and two CEPs on the MediaVideo Player entry-points A3 and A6. The Rover capsules use NOS information G_{EPT}, and M_{QON} and its internal knowledge of $S_{ROS}()$ and $S_{MAP}()$ (for this simple example we assume that it is given in Table-3) to determine the best placement of the transcoders. Fig-8 shows the optimum placements of the "Xcoder" units for this AVT

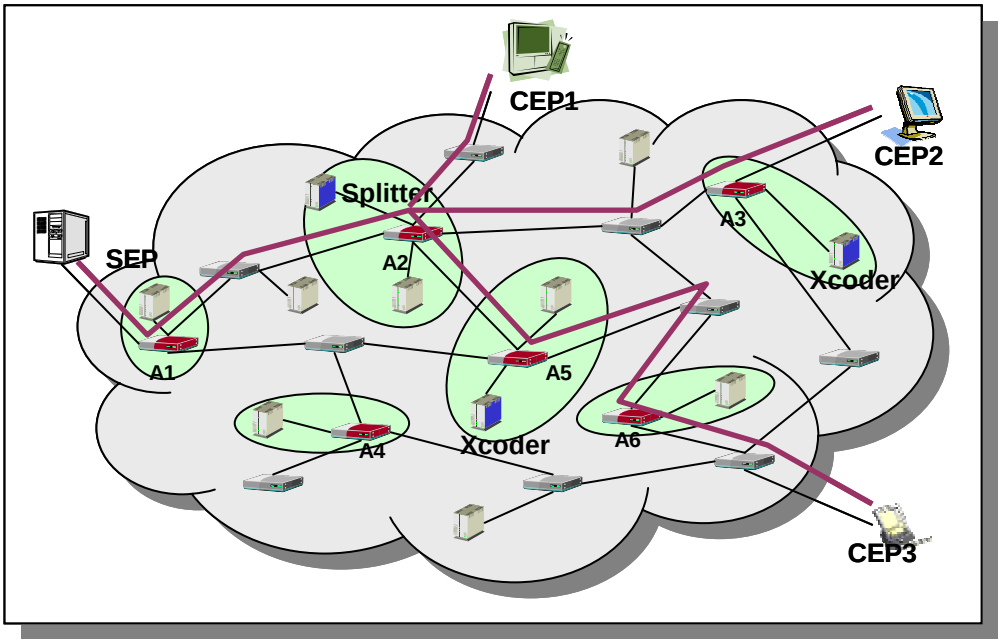

Fig-8. The base topology between one server and three players. Nodes A2, A3 A5 maps one splitter and two transcoders respectively.

channel instance created for MediaVideo. Once the "Xcoders" are activated, SEP and CEPs return handles to its MPEG2-stream ports to the corresponding MediaVideo end-points-- again via the EPPI interface.

4.6.3 Execution

Any stream sent/received via the port now receives automatic distribution and embedded rate adaptation. The subscriber operation interacts with the channel only via SEP and CEP end-points. The channel designer provides documented sending and receiving interfaces under EPPI. During it normal streaming operation NOS does not get involved. If underlying node/link traffic dynamics changes, then this is tracked by the periodic instantiation of the Rovers. Also, if a new endpoint attempts to join in or leave then accordingly a CEP is invoked or terminated, and the corresponding change is eventually caught by the Rovers.

5. Conclusions

In this paper, we have discussed the requirements for an advanced framework that can support development of complex netcentric applications for active network. We have also outlined the Active Channel Framework[4] that can support many of these requirements. ACF provides formalism 'components' and 'interaction environments' for both the channel developer and the subscriber application entities. The complexities arising from these distinctions did not arise in previous active node programming models.

We have illustrated this construction formalism by using an example concept application-- a nomadic

[4] Some researchers have proposed a three-tier active network architecture based on the abstraction of Node OS, EE, and AA [26,28]. It will be difficult to map these with the tiers of ACF. Unlike these efforts which started from routers architecture, ACF is the result of a top-down investigation on multiparty active system development 'framework' (rather than on the 'architecture'). It has more than one notion of 'environments'. The 'interaction environments' are distinctly different with non overlapping characteristics. Also the meaning of 'programming' is separate at two tiers. Since, the definitions are still open and evolving, the comparison attempt will remain open ended.

MPEG-2 rate transcoding mechanism. Within the scope of this paper, however, we have not elaborated on the detail of the transcoding operation. More information about the SONET (Self-Organizing Network Embedded Transcoder) system can be found in [6,15].

In our framework design we emphasized reusability, and task division. We placed core services in the network layer needed for supporting nomadic and self-organization behavior of adaptive systems. Additionally, we have introduced the concept of programmable channel construct to encourage reusability, instead of building the video transmission specific adaptive logic directly into the application core. Here the service is the video rate adaptation. The service itself although requires complex organization and design, but apparently it can be used by any video application with simple socket like interface.

Though we have presented the case with the transcoder channel, but this framework is the result of our experience with a number of active applications (active prefetch-proxy [17], daisy-chain forwarder [14], Active Harness group communicationware[16], etc). Nevertheless the applications are further being expanded. For example, we have recently demonstrated transcoding channel which performs the transcoding operation by distributing the computation over an active subnet, instead on a single active node. This extension did not require any significant modification of the formalism as its mapping and dynamic relocation capabilities were already sufficient to support such distributed computation.

An interesting issue is scalability of active systems. A particular concern in any complex and large application is the complexity of the signaling mechanism. We think there is nothing which will prevent a good designer to build schemes which will scale. For example, let us take a closer look into the communication patterns of the Rovers. The given Rover algorithm analyzes the base network to determine the placement of the transcoders and their step-down ratios. This is the potential bottleneck point which involves activity in the largest part of the network in the ACF. Yet an analysis will reveal that this entire scheme is amazingly light. It uses only two signal messages (request/grant) per link for this entire job. This O(1) cost does not depend on the size of the network. This scalability has been derived by avoiding any duplicate information from being propagated. All information was optimally pruned at the forwarding nodes (in this case the min/max rules). Indeed, this is one of the advantages of embedded programmability. Information can be collected in distributed scalable way when needed. Therefore in ACF there is no centralized M_{QON}. All its quantities are probed locally. In general the network embedded processing ability will increase the designer's flexibility to build scalable solutions. However, there is noting which will prevent a bad designer either. Also, there might be application specific task segments which are inherently non-scalable. It suffices to say that scalability of active systems can not probably be meaningfully discussed without knowing the active application. The ACF approach is to empower channel programmer to exploit it maximally where it exists.

Active and programmable network is a relatively new area in networking research. The framework of systems development on such network is perhaps one of the least explored yet enormously challenging areas within it. ACF is one of first explorations to analyze network programmability from the software engineering with several serious applications scenarios. We have raised few of the issues. Clearly we expect its individual components to advance. Programming active systems will be much more complex than end-to-end systems. There is lesser doubt in its capability to add tremendous value in communication services. The bigger challenge however, lies in the techniques of complexity management.

The work is currently being funded by the DARPA Research Grant F30602-99-1-0515 under its Active Network initiative.

6. References:

[1] Amir, Elan, Steven McCanne, and Randy Katz, Receiver-driven Bandwidth Adaptation for Light-weight Sessions, Proceedings of ACM Multimedia ' 97, Seattle, WA, Nov 1997,

[2] Bhattacharjee, S., Kenneth L. Calvert and Ellen W. Zegura. An Architecture for Active Networking. High Performance Networking' 97 ,White Plains, NY, April 1997. [also available at http://www.cc.gatech.edu/projects/canes/papers/anarch.ps.gz, October 98]

[3] Fluckiger, F, Understanding Networked Multimedia Applications and Technology, Prentice Hall, UK, 1995.

[4] Haskell B. G., Atul Puri and Arun Netravali, Digital Video: An Introduction to MPEG-2, Chapman and Hall, NY, 1997.

[5] Information Technology- Generic Coding of Moving Pictures and Associated Audio Information: Video, ISO/IEC International Standard 13818-2, June 1996.

[6] Javed I. Khan, S. S. Yang, Medianet Active Switch Architecture, Technical Report: 2000-01-02, Kent State University, [available at URL http://medianet.kent. edu/ technicalreports.html, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet]

[7] Keesman, Gertjan; Hellinghuizen, Robert; Hoeksema, Fokke; Heideman, Geert, Transcoding of MPEG bitstreams Signal Processing: Image Communication, Volume: 8, Issue: 6, pp. 481-500, September 1996,

[8] Javed I. Khan, Motion Vector Prediction in Interactive 3D Video Stream, Proceedings of the World Congress on Advanced IT Tools, IFIP '96 IT, Canberra, Sept 96 , pp533-539.

[9] Tennenhouse, D. L., J. Smith, D. Sincoskie, D. Wetherall & G. Minden., "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, Jan 97, pp 80-86

[10] Wetherall, Guttag, Tennenhouse, "ANTS: A Tool kit for Building and Dynamically Deploying Network Protocols", IEEE OPENARCH'98, San Francisco, April 1998. Available at: http://www.tns.lcs.mit.edu/publications/openarch98.html

[11] Youn, J, M.T. Sun, and J. Xin, "Video Transcoder Architectures for Bit Rate Scaling of H.263 Bit Streams," will beappeared to 'ACM Multimedia 1999', Nov., 1999. pp243-250.

[12] Jill M. Boyce and Robert D. Gaglianello, Packet loss effects on MPEG video sent over the public Internet; Proceedings of the 6th ACM international conference on Multimedia , 1998, Pages 181 - 190

[13] White Paper, Delivering RealAudio® or RealVideo® from a Web Server, RealNetworks Technical Blueprint Series, 1998 [URL:http://service.real.com/help/ library/blueprints/ hosthtml/webhost.htm Last Download: Sept 20, 2000]

[14] Javed I. Khan, & S. S. Yang, Made-To-Order Custom Channels for Netcentric Applications over Active Network, Proc. Of the Conf. On Internet and Multimedia Systems and Applications, IMSA 2000, Nov 2000, Las Vegas, pp22-26.

[15] Javed I. Khan, Seung Su Yang, Qiong Gu, Darsan Patel, Patrick Mail, Oleg Komogortsev, Wansik Oh, and Zhong Guo Resource Adaptive Netcentric Systems: A case Study with SONET- a Self-Organizing Network Embedded Transcoder, Proc. of the ACM Multimedia , Oct. 2001, Ottawa, Canada, pp.617-620.

[16] Javed I. Khan and Asrar Haque, An Active Programmable Communication Harness for Measurement of Composite Network States, Submitted to IEEE International Conference on Networking, ICN' 2001, pp628-638.

[17] Javed I. Khan and Qingping Tao, Partial Prefetch for Faster Surfing in Composite Hypermedia, 3rd USENIX Symposium on Internet Technologies and Systems, USITS 2001, San Francisco, March 2001, pp13-24.

[18] Coulouris, G, and Dollimore, J, Distributed Systems: concepts and Design, Addison Wesley, 2nd ed., 1994.

[19] Corbin, J. R., The Art of Distributed Applications, Programming Techniques and Remote procedure Calls, Springler-Verlag, 1991.

[20] Siegel, J., CORBA Fundamentals and programming, Wiley, 1996.

[21] Otte, R., Patrick, P, Roy, M, Understanding CORBA, prentice Hall, 1996.

[22] Gopalan Suresh Raj, A Detailed Comparison of CORBA, DCOM and Java/RMI, [URL: http://www.execpc.com /~gopalan/ misc/compare.html,retrieved on Dec. 1999]

[23] Plug-in Guide for Netscape Communicator, Netscape Communications Corporation, 1997, [URL: http://developer.netscape.com/docs/ manuals/communicator/plugin /index.htm, Last Retrieved October 2000]

[24] Javed I. Khan, Q. Gu, Darsan Patel and Oleg Komogortsev, , Medianet Active Video Transcoder Performance, Technical Report: 2000-01-02, Kent State University, [available at URL http://medianet.kent. edu/ technicalreports.html, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet]

[25] Lee, Edward, What's Ahead for Embedded Software, IEEE Computers, Vol. 33, No. 9, September 2000, pp-18-26.

[26] Xuebin Xu, "A Survey of Ideas in Programmable Networks", Technical Report: 2002-01-01, Kent State University, [available at URL http://medianet.kent.edu/ technicalreports. html, also mirrored at http:// bristi.facnet.mcs.kent.edu/ medianet]

[27] Andrew T.Campbell, etc., "A Survey of Programmable Networks" Computer Communication Review, vol.29, no.2, pp.7-23, Apr.1999.

[28] Thomas M.Chen, etc., "Active and Programmable Networks", guest editorial for Special Issue: Active and Programmable Networks, Network Interactive, May 1998.

[29] Gary Stix, The Triumph of the Light, Scientific American, January 2001, pp31-35 [HTTP:// http://www.sciam.com/2001/0101issue/0101stix.html]

[31] Beverly Schwartz, Wenyi Zhou, Alden W. Jackson, W. Timothy Strayer, Dennis Rockwell, Smart Packets for Active Networks. In 2nd Conf. on Open Architectures and Net-work Programming, OPENARCH'99,NY, Mar. 1999. [URL: http://www.ir.bbn.com/~bschwart/, Last retrieved 10//02/00]

[32] M. Hicks et al. PLANnet: An Active Internetwork. In Conf. on Computer Communications, INFOCOM'99, pages 1124–1133, New York, NY, Mar. 1999. IEEE.

[32] Y. Yemini and S. da Silva. Towards Programmable Networks. In Intl. Work. on Dist. Systems Operations and Management, Italy, Oct. 1996. [URL: http://www.cs.columbia.edu/dcc/netscript/Publications/publications.html, Last retrieved: 11/02/00]

[33] Peter Forman and Robert W. Saint, Creating Convergence, Scientific American, November 2000, pp.10-15[HTTP://http://www.sciam.com/2001/0101issue/0101stix.html]

[34] Y. Yemini and S. Trito, Nestor: Technologies and protocols for self-managed and self-organizing networks. URL: http:// www.cs.columbia.edu /dcc/nestor, 1998.