Proceeding of the International Conference on
Internet and Multimedia Systems and Applications, IMSA 2000.
November 20-23, 2000 Las Vegas, USA, pp 280-285.

MADE-TO-ORDER CUSTOM CHANNELS FOR NETCENTRIC APPLICATIONS OVER THE ACTIVE NETWORK

JAVED I. KHAN & 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

In this paper we present a formalism, which might enable building programmable or soft composable custom channel construct for complex net centric applications, instead of the standard service stack that is available today.

Key words: active network, composable channels, netcentric applications.

1. Introduction

Low-level network architecture has enormously advanced through rapid innovations in recent years. However, as far as applications are concerned, the only service model and the conduit to these advancements available to them to date is the TCP or UDP socket (and their minor variants) [1]. While they are proven tool for many traditional applications, there are also already many modern network applications for which neither seems to be quite suitable [2,3,4]. They require more sophisticated and customized communication service from the underlying network. For those service variants—if these are at all realizable, applications have to recompose them grounds' up-- often reinventing wheels.

In this research, however, we do not propose to create yet another 'perfect' service interface, rather we propose the infusion of programmability into network where custom soft composable channels can be built for fulfilling the varying but specific needs of the applications—we call it *made-to-order channels* (MTO channel) It seems the fundamental problem is not with the TCP or UDP in particular— but any single such standard service stack will continue to lose cure-for-all appeal as applications are leaping forward to build enormously complex distributed systems. It seems a better solution may lie in increased programmability inside network.

In this paper, we particularly show a mechanism that allows building reusable and soft composable custom channels. The formalism allows building qualitatively a new breed of sockets-- where channels also provide interactivity and information exchange with application end-points enabling dynamic exchange of local states between network and the applications. Potentially, not only a new class of applications but also a new class of solutions to many of the current hard-to-tackle network layer problems can be found with this formalism [5,6,7]. The concept is based on another novel idea that has emerged over last few years—active networking.

2. Related Work

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 DCOM/COM, CORBA, Java/RMI) [8,15]. More recently, active networking is exploring the means for adding programmability into network data path. At least eight formalisms have been explored. Examples include SmartPacket [9], ANTS [10], PLAN [12], NetScript [13]. The distinguishing aspects of this work are twofold. First, most of the previous works have eventually focused on building more complex process construct. In contrast here we focus on the composition of the complex channel construct. Nevertheless the work is a more concrete formalization of the vision pursued under active networking [7,11]. The other novel aspect is the bi-direction interactivity between the channel and the application.

We have designed an active network based made-to-order custom channel *building and execution* system. Within the scope of this paper we only present a high-level design description of this concept system. Section 3 first defines the notion. Section 4 then describes the formalism and the mechanism for its composition and execution. Finally, in section 5 we explain the above steps by way of an example channel—that adds a fundamentally new service in network layer—storage.

3. Concept: Made-to-order Channel

A channel provides a coordinated set of communication services to a patron application for sharing information between it's two or more end-points.

Proceeding of the International Conference on Internet and Multimedia Systems and Applications, IMSA 2000.
November 20-23, 2000 Las Vegas, USA, pp 280-285.

Each channel handles one principle information stream. A channel itself generally has one source-end, one sink-end, and possibly a set of embedded network programs. The generalized channel that we propose extends the classical notion of channel in the following ways:

- **Programmability:** Programmability refers to the service model where the components of the channels are programmable. A channel designer can design and make it available to others. Patrons should be able to further configure it by adjustable configuration variables. Installation and its use should not require low-level administrative privilege so far it conforms to the design formalism.

- **Event Notification:** Channels should have a mechanism to report designated events to patrons. A patron should be able to switch on or off notification features. When ON, the end-points should be able to receive interrupt by event handler routines.

- **Status Polling:** An application should be able to read and write designated status variables in a capsule. Capsules should also have access to a set of local network states.

It will be up-to the channel designer to define the configuration options, status variables, and events. It will also be up to the channel designer to set the read/write privileges on these variables and to specify the trap and trigger privileges for events. In this paper we limit the discussion for single source single sink channels.

4. System Architecture

4.1. Virtual Switch Machine

The made-to-order channel system has been built on **Medianet Active Switch System** developed at Kent [14]. The base of this two-tier system is the **Virtual Switch Machine** (VSM). It provides the three basic services--stream interception (or switching), module execution, and the routing over the hardware. It intercepts streams and inserts code modules (*switchlets*) in the flow path of a data stream. The system also loads and unloads the *switchlets* at network junction points and resolves critical resource contention. The current system manages

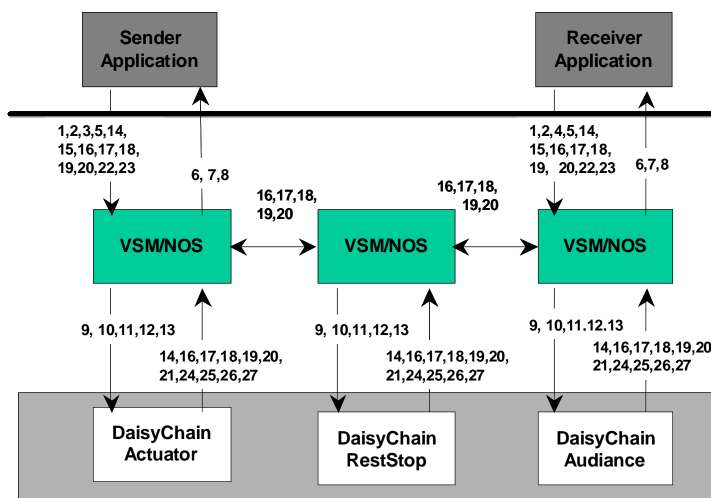

Fig-1 System Architecture and Communication

contention for e*xecution cycles*, *switchlet session memory*, and *forwarding order*. It allocated but does not actively resolve contention for ports and local file space.

Capsule Set	VSM/BEE Set
9 : CAP_Initialize 10 : CAP_NewStream 11 : CAP_TriggerEvent 12 : CAP_DestroyStream 13 : CAP_Destroy	14 : NOS_RegisterEvent 15 : NOS_GetEventList 16 : NOS_GetStatus 17 : NOS_SetStatus 18 : NOS_TriggerSend 19 : NOS_TriggerRecv 20 : NOS_TriggerEvent 21 : NOS_GetNetStatus 22 : NOS_EnableEvent 23 : NOS_DisableEvent 24 : NOS_GetLocalNetworkEventValue 25 : NOS_SetLocalNetworkEventValue 26 : NOS_GetLocalNetworkStatus 27 : NOS_SetLocalNetworkStatus
Patron Set	
1 : EP_OpenTransponder 2 : EP_CloseTransponder 3 : EP_Bind 4 : EP_Connect 5 : EP_Close 6 : EP_TriggerEvent 7 : EP_TriggerSend 8 : EP_TriggerRecv	

Table 1

4.2. Builder & Execution Environment

The application or channel programs are not intended to have raw access to VSM services and resources. Rather the code modules of channels have to follow additional descipline. Also, a set of utility is provided. This is done by a second abstraction layer called **Channel Building & Execution Environment** (Channel BEE). The system operates through three sets of standard functions. The first set is for the capsules (NOS Service). The second set is for the Applications (EP service). However, unlike other formalisms—there is also a third set provided by the capsules (CP Service) to be invoked by the VSM/BEE. NOS set also includes utility functions for receiving local network states from VSM. Table-1 shows the sets and Fig-1 illustrates the involved entities and their direction of interactivity.

4.3. Channels

Description of each channel is called a channel *forma*. A forma has the codes (or URL pointers) for the *capsules* a *port table* describing the connection matrix connecting the *capsules* and a *deployment map*, specifying the deployment plan for the modules. Each channel has one *actuator* end-point capsule, an *audience* end-point capsule and zero or more *transponder* capsules. For each instance of a channel generally there should be one instance of the actuator, one audience, and zero or more instances of transponders. The physical addresses of

the capsules are not available prior to the channel deployment. The forma uses platform independent *Channel Relative Addressing* **(CRA)** language for specifying locations and addressing each other.

4.4. Channel Relative Addressing

It provides the capsules and patron applications an abstraction to reference each other without knowing the physical casting of the components. All addresses are described relative to their own position, source and sink. It supports addresses such as: *this-node, source, sink, all, any, none, all up/down-stream, any up/down-stream, m up/down-stream, nth-up/downstream, etc.* For multicast tree it also has addresses such as *joint, m-way-joint, any-split, m-way-split*, and their *all, any, upstream, downstream* variants. The CRA also allows address algebra such as *((source).3rd-dowstream).all-up-stream)* in this addressing scheme.

4.5. Channel Capsules

The channel capsules are the principle modules where the channel designer declares the local event set and the local status variables. It also contains the logic that computes them and the propagation rules. Channel designer also assigns the access level (set or get) and scope (local to capsule, local to channel, accessible to patron etc.) The *Actuator* and the *Audience* capsules have the additional responsibility to manage the events and status variables accessible to the respective patron end-points.

4.6. Channel Building and Execution

When the channel is created, the VSM/BEE obtains the *forma* and the layout plan and the capsules pointed in it. When the first application end-point binds the *actuator* endpoint is invoked and the channel is compiled. At compilation stage—resources are allocated only at the end-point of the requestor. When, the other application requests connection, the *audience* end-point is then invoked and the actual channel construction begins. The first step is the determination of the *casting plan*. There are two phases—first the CRA addresses are translated into active node locations. The second step is the port allocation for all the logical sub-channels those interconnects capsules. A request is sent out to the VSM/BEE from the actuator node requesting the specific number of ports. When the port addresses are back then the actual *casting plan* is computed and the deployment begins. In this stage the channel builder creates packets full of capsule and connection information and sends them to the right active nodes. The target node's VSM/BEE performs security and local consistency verifications and allocates resources to the capsules.

The channel runs when the patron end-points attach 'information' units to the made-to-order socket end-points and then trigger the flow (in send or receive mode).

Daisy Chain Rest Stops

```
(a) CAP_Initialize()
{
   Event timed_out, return_receipt, sweep,
   change_dead_line, consumer;
   Event ForwardLoadChange, BackwardLoadChange;
   Event consume;

   // register events
   time_out.name = "time_out";
   time_out.type = USER_RD | USER_NOTIFY |
   CAPSULE_RDWR | CAPSULE_NOTIFY;
   NOS_RegisterEvent(time_out);
   ….
   sweep.name = "sweep";
   sweep.type = USER_RDWR | USER_NOTIFY |
   CAPSULE_RD | CAPSULE_NOTITY;
   NOS_RegisterEvent(sweep);
   consume.name = "consume";
   consume.type = USER_NORDWR | USER_NONOTIFY |
   CAPSULE_RDWR | CAPSULE_NOTITY;
   NOS_RegisterEvent(consume);
   ….

   // set event handler

   return_receipt.handler =     CAP_ReturnReceiptHandler;
   sweep.handler = CAP_SweepHandler;
   consume.handler = CAP_ConsumeHandler;
   ….

   NOS_SetEventValue(return_receipt);
   NOS_SetEventValue(sweep);
   NOS_SetEventValue(consume);
   …..
   ……
   // get event of network load monitor and set the event handler
   BackwardLoadChange.name =
"backward_link_load_change";
   NOS_GetLocalNetworkEventValue(BackwardLoadChange);
   BackwardLoadChange.handler
      = CAP_BackwardLinkLoadLevelChangeHandler;
   NOS_SetLocalNetworkEventValue(BackwardLoadChange);
}
```

```
(b) CAP_ImmediateSweepHandler()
{
   // Get load_threshold Status and change value
   NOS_GetStatus(load_threshold);
   load_threshold.value = HIGH;
   // invoke sender routine
   NOS_TriggerEvent(consume);
   NOS_SetStatus(load_threshold);
   sweep.dst = PrevHop;
   // Propagates immediate_sweep event
   NOS_TriggerEvent(sweep);
}
```

Fig-2 Daisy Chain Rest Stop Modules

Patron end points can query for a list of events and status available from a made-to-order channel and can subscribe to individual events by registering handlers. Either the actuator or the audience capsule handles all patron requests. The BEE acts as the mediator in event propagation and status polling and VSM provides the resource scheduling and execution.

5. Example Patron Application

Any net-centric video application offers a scenario rich in complex synchronization and communication service requirements. Video distribution itself will benefit from custom channels with abilities for in-stream transcoding, time routing, automatic macro-block healing (as opposed to retransmission), and rapid impairment notification (when something goes wrong in network), automatic minimum impact cache replenishment, etc. Also a channel that can provide notification and status information such as running statistics about congestion level, timeliness of delivery, rate conversion and video quality statistics can be very useful. To explain the process of the MTO channel construction here we present a novel made-to-order channel named as *Daisy Chain Replenishment Channel (DCR).*

5.1. Daisy Chain Replenishment Channel

This channel is intended for carrying massive video feed from the video distribution center (Origin Server) to the strategically located Cache Servers via long haul shared network during off-peak hours of the hops. However, all the hops may not have the off-peak simultaneously. The idea is to install a series of file stops at selected network points inside a long haul network. If a link is busy, it should automatically store the transit traffic in a secondary buffer (such as hard drive). When the network load eases, it should then forward the waiting data to the next stop. As evident, synchronization is dependent on local network state. Our purpose is that application end-points should not have to worry about all the detail synchronization needed to implement this. As far as application end-points are concerned, they should simply be able subscribe and install the DCR-channel, specify a delivery schedule and be able to get this service.

We have also specified additional smart interactive features. The channel provides a number of status and event services. For example, when, the channel anticipates that a delivery may not meet the schedule based on state of network, it can send notification to the receiver application. It can also send some 'reasons' that why and where it’s is expecting failure. Job completion itself is another event. Also if requested, the channel can report the job status, such as how much of the requested data has been transferred up-to which point and what is the expected delivery time. It can also accept a 'sweep' signal, where the patron (in this case we assume a receiver) can change its mind (such as an user has requested a video from the cache) and request all pending data to be delivered immediately.

5.2. Channel Construction

The channel has three capsules--DaisyChainActuator, DaisyChainRestStops and DaisyChainAudiance, and a *port tabl*e. Its events include “timed_out”, and “return_receipt”, and “sweep”. It has read only status variable “travel_profile”, and a write-able status variable “load-threshold”. (when line load is below it a capsule sends file to the next hop). Fig-2 provides part of the pseudo-code for the DaisyChainRestStop It has the standard threads CAP_Initialize(), CAP_Destroy(), and the event handlers. The CAP_Initialize() (Fig-2 (a)) first registers the event variables and their attributes. For example, the event ‘sweep’ is a user triggarable event. It also then registers the event handler routines including those for low-level network state events. Always CAP_Initialize() is invoked by VSM/BEE right after instantiation of the capsule.

(a) Origin Server (Sender) Application

```
Initialize()
{
   EP_OpenTransponder(cd, DaisyChain);

   NOS_GetEventList(&count, &evl);
   ….
   time_out.name = “time_out”;
   NOS_EnableEvent(time_out);
   return_receipt.name = “return_receipt”;
   NOS_EnableEvent(return_receipt);
   timed_out.handler = TimeOutHandler;
   return_receipt.handler = ReturnReceiptHandler;
   NOS_SetEventValue(timed_out);
   NOS_SetEventValue(return_receipt);

   LocalAddr = localhost;
   EP_Bind(cd, LocalAddr);
}
…..
```

(b) Cache Server (Receiver) Application

```
Initialize()
{
   ….
   EP_OpenTransponder(cd, DaisyChain);
   …
   RemoteAddr = remotehost;
   EP_Connect(cd, RemoteAddr);
   …..
}

ImmediateSweep()
{
   sweep.dst = PrevHop;
   NOS_TriggerEvent(sweep);
}
```

Fig-3(a) Origin and (b) Cache Server End Points

The principal routine that handles the data communication are simple producer/ consumer threads (not shown). The producer thread receives file segments if the buffer is not full. The consumer thread is triggered when load level change is detected, threshold is changed or a previous send ends and the buffer is not empty. It uses an internal event trigger called “consume” for that.

The thread CAP_ImmediateSweephandler() (Fig-2(b)) is invoked if a sweep event is detected. It changes the status variable “load-threshold” triggering immediate transfer of all pending data. It also propagates the sweep event to the capsule immediately following it.

The DaisyChainActuator and DaisyChainAudiance (not shown) are similar, but they have few additional responsibilities to serve the upper level applications. Thus, when a status poll set or get request is made, or an event is triggered by the patron, they propagate them to the right internal capsules. When the Origin Server (or Cache Server) sets a new deadline, DaisyChainActuator (or DaisyChainAudiance) calls CP_SetStatus().

Finally we show the application end-points. Fig-3(b) illustrates the Origin Server. It first requests the custom channel “DaisyChain”. Then it queries about the avilable event and status variables. It then selectively subscribes to the events it is interested in by bindling its own handler to the subscribed events. If an event is not subscribed it is by default disabled. Similarly, the CacheServer (Fig-3(b)) connects to the actuator. Also we show the code that can be used to trigger a sweep. It can simply trigger the event. VSM/BEE carries the trigger to the audience capsule. Accordingly, CAP_ImmediateSweepHandler() is invoked. It changes the threshold to a high value to enable a local sweep and then propagates the event in the backward direction as defined in this capsule (Fig-2).

6. Conclusions

In this research we have tried to demonstrate the feasibility of a novel mechanism which can offer a technique for composition of novel channels beyond classical TCP or UDP. It also introduces the concept of interactive channels. We have illustrated the design and implementation formalism by using the example of a smart channel that injects a new service in network layer—storage. More details of the codes are in [16].

Further research has to be carried out in a wide range of associated issues including software engineering, secured interaction between channels on active nodes, node resource allocation.

Runing soft states is intrinsically expensive. However, the ability to interplay local network and application states provides fundamental qualitatively advantage. Therefore active channels can not be easily compared with conventional switced ones. More critical is the cost of diversion operation in a potential hybrid active/passive switch. This may not add too much overhead in the data path of the conventional switcing. Also, in this work we have limited scope for single source and single end channels. More general 1-n multi-cast or m-n multi-source made-to-order channels are also conceivable.

The work is currently being funded by DARPA Research Grant F30602-99-1-0515 under it's Active Network initiative. The authors would also like to acknowledge the support of Ohio Boards of Regents.

7. References

[1] Comer D. E., Internetworking with TCP/IP, Principles, Protocols, and Architectures, 4th Ed, Prentice Hall, New Jersey, USA, ISBN- 0-13-018380-6, 2000

[2] E. Amir et al. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proc. SIGCOMM'9*8, pages 178–189. ACM, Sep 1998.

[3] E. Johnson. A Protocol for Network Level Caching. M.Eng Thesis, MIT, May 1998.

[4] C. Papadopoulos et al. An Error Control Scheme for Large-Scale Multicast Applications. In Conf. on Computer Communications (INFOCOM'98),San Francisco, CA, April 1998.

[5] N. C. Hutchinson and L. L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. IEEE Trans. on Software Engineering, 17(1):64–76, Jan. 1991.

[6] K. Ramakrishnan and S. Floyd. A Proposal to add Ex-plicit Congestion Notification (ECN) to IP. Request For Comments 2481, IETF, Jan. 1999.

[7] Wetherall, David, Active Network Vision and Reality: Lessons from capsule-based System, Operating Systems Review, 34(5):64-79, December 1999.

[8] 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]

[9] B. Schwartz et al. Smart Packets for Active Networks. In 2nd Conf. on Open Architectures and Net-work Programming, OPENARCH'99,NY, Mar. 1999. IEEE.

[10] D. Wetherall et al. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In 1st Conf. on Open Architectures and Network Programming OPENARCH'98, pages 117–129, San Francisco, CA, Apr 1998. IEEE.

[11] 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

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

[13] [Y. Yemini and S. da Silva. Towards Programmable Networks. In Intl. Work. on Dist. Systems Operations and Management, Italy, Oct. 1996.

[14] 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]

[15] 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]

[16] Javed I. Khan & S. S. Yang, Custom Channels: Daisy Chain Replenishment, Technical Report: 2000-10-02, Kent State University, [available at URL http://medianet. kent.edu/technicalreports.html, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet]