Enhancing the Touring Machine API to Support Integrated Digital Transport

Mauricio Arango, Michael Kramer, Steven L. Rohall, Lillian Ruston, Abel Weinrib

Bellcore, 445 South Street, Morristown, NJ 07962-1910

Abstract. The current version of the Touring Machine^{TM} software platform for multimedia communications uses analog transport of voice and video media streams. The next iterations of system design will, among other enhancements, support digital (packet-switched) transport of all media streams, placing new requirements on the system. This paper describes proposed changes to the Touring Machine Application Programming Interface in support of integrated digital transport. We identify new abstractions in two areas: network access control and specification of transport topology. These new abstractions will provide flexible control of multiple streams sharing the underlying integrated transport (for instance, separate packet-video streams each being displayed in separate windows on a multimedia workstation) and will support heterogeneous terminal equipment and networks by transparently converting media-stream formats within the network.

1 Introduction

The Touring Machine project at Bellcore is concerned with developing a research software platform to support multimedia communications applications, with the goal of learning about key technical questions important to realizing a public network infrastructure to support such applications. The Touring Machine software infrastructure provides to application programmers an Application Programming Interface (API) that facilitates the development of multimedia communications applications by supporting a rich set of abstract capabilities of the system. The project includes an experimental multimedia communications testbed composed of a network of desk-top video and audio devices and workstation-based shared workspaces controlled via users' workstations. The testbed provides the infrastructure for communication tools used daily by 150 users in two Bellcore locations 50 miles apart; these tools include multimedia, multipoint conferencing and information services as well as point-to-point communications. For a more complete description of the Touring Machine project and API, see, e.g., [1].

The current version of the Touring Machine software is the second iteration of system design. It uses analog transport to provide the voice and video media, controlling analog audio and video switches and other specialized hardware devices (such as bridges to create multi-party communications sessions) which

^{TM} Touring Machine is a trademark of Bellcore.

connect monitors, cameras, microphones, and speakers on the users' desk tops. The signaling messages between applications and the Touring Machine infrastructure, and between different components of the Touring Machine software architecture, are carried "digitally" on our local internet, as is the "data" media type used, for example, to transport text or X Window System^{TM} protocol messages. We are presently in the design phase for a third version of Touring Machine, which will include many enhancements, among them support for integrated digital transport for the different media streams controlled by the system. The choice of off-the-shelf analog voice and video hardware for the current version has allowed us to support a significant user community, but in the next version we want to broaden the transport technologies we support.

Moving from analog to digital transport places a number of new requirements on the system and raises many interesting questions. This paper focuses on some of the changes to the Touring Machine API and its underlying abstractions required to support a hybrid analog and digital network infrastructure. In *digital* we include packet-switched transport that supports dynamic bandwidth allocation as well as circuit-switched transport such as ISDN; by *hybrid*, we mean that the system will support heterogeneous transport of media (analog as well as different formats of digital) providing an integrated, media-format-independent set of abstractions across the API. A hybrid system will transparently convert between different media formats when needed to facilitate communications between terminal equipment with different capabilities. Thus, a single *terminal* can comprise a mixture of both analog equipment and workstation-supported digital audio and video, allowing an application the flexibility to, for example, display a video stream wherever the user prefers. The goal is for an application programmer, using the API, to be able to write an application independently of the specific hardware that will be used, whether analog or digital.

2 API Changes

The current Touring Machine API separates the control of media streams into two components: network access control via mapping of logical endpoints to physical ports, and definition of a transport topology connecting the logical endpoints. This separation is viewed by programmers who have created applications to run on the Touring Machine system as one of the strengths of the API because, for example, it supports management of, and sharing of network access resources between, multiple concurrent multi-party sessions; we want to preserve this separation when moving to digital transport. We first discuss the new network access abstractions of the Touring Machine API that allow an application programmer to take advantage of the flexibility of integrated digital access to the Touring Machine controlled network. Then, we will turn to specifying the transport topology.

^{TM} X Window System is a trademark of MIT.

2.1 Network Access

In the current analog-only system, network access is described by ports, where each port provides access for a single stream of a given media type to the system. In our analog system, the port abstraction is appropriate: a port corresponds to a cable that connects desk-top equipment to the Touring Machine-controlled network, such as a coax cable used to carry a single stream of NTSC video that terminates on a video switch within the network. The ports are static, changing only when a new cable is installed. The current port abstraction could be applied to a digital network which emulates circuit-based network access using pre-established virtual circuits reserved to carry only a single media stream of a single type. However, this approach would not allow the application programmer to utilize the flexibility of integrated digital transport, where a more dynamic port abstraction is appropriate.

With digital transport, the same network access link (e.g., an Ethernet^{TM} tap, or a fiber carrying ATM cells) can be used to carry a variety of different information streams. Thus, we extend the current network access abstractions to include two new types of objects in addition to ports: access links and access channels; see Figure 1.

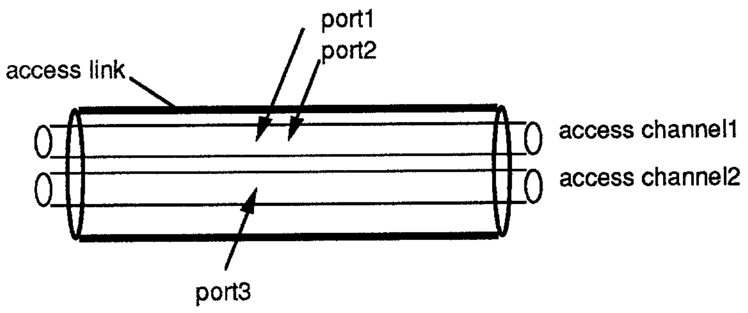

Fig. 1. Links, channels and ports.

An access link provides "raw" connection into the network, for instance a fiber carrying ATM cells, and is relatively static, changing only when a terminal's access into the system is modified. A link is described by the basic transport it provides (e.g., ATM cells or NTSC analog video) and the link resources (for instance, bandwidth) it contains. While an access link may correspond to a single physical connection into the network, the link abstraction is more general: a link may also correspond to a portion of a fiber's bandwidth set aside to carry ATM cells, or to a set of coax cables that all carry analog video.

^{TM} Ethernet is a trademark of Xerox.

An access channel is created on a particular access link; its type is the same as that of the link. Creation of a channel reserves a subset of the link resources (bandwidth, etc.), and fails if there are not enough resources available. The channel abstraction provides a mechanism to the application programmer for managing separate portions of the bandwidth on a link, thus supporting quality of service guarantees for multiple applications (perhaps serving different users) on a single desk-top terminal. This abstraction also gives the application programmer explicit mechanisms for multiplexing different streams into limited portions of the total link bandwidth. (Few available packet-network technologies support the bandwidth reservation and "fire walls" required to fully implement channels at this time, but work in this area is active. Furthermore, the channel abstraction can be useful even if only to control link usage in software.) After creation, the channel can be deleted or modified to change the link resources allocated to the channel.

Once channels have been created, an application can create a typed port on a particular channel. The type of a port fully specifies the media stream that is going to be transported over the channel. The port type must be consistent with the type of the channel, and serves as a complete description of everything that the system needs to know to transport the stream. Thus, for example, attempting to create an analog video port on an IP packet network will fail, while a fully specified video port that could be successfully created might be "JPEG-encoded video with frame rate f and quality parameter q in IP packets, UDP transport protocol." The specification of the port is used for two purposes: it describes to the network how to process the media stream (including quality of service requirements), and it can be used by another end-terminal to understand the type of the media stream.

A port type must minimally describe the bandwidth and other quality of service requirement needed to provide transport. However, specifying other parameters (such as media type) is necessary for the use of other infrastructure services beyond simple transport. For instance, the port must specify the media type and encoding information if the system is to support communication between incompatible terminals by converting between different media encodings. This flexibility in the amount of information that is described by a port type allows the system to support basic transport of media streams of any type, as well as more complex processing of streams of specific supported types.

While a logical association is created when a port is created on a channel, none of the channel resources are dedicated to the port at that time. A port simply specifies typing information and provides a "handle" for the media stream that will eventually be carried; thus, a port is in some ways analogous to a UNIX[TM] socket. The channel resources required to transport the stream are only allocated to the port later in the process when a logical endpoint is mapped to the port, making it *active*. (An active port is actually transporting a media stream.) Many ports may be active on a channel, constrained only by the total channel resources available.

[TM] UNIX is a registered trademark of Unix System Laboratory, Inc.

The separation of access link resource reservation (using channels) from specification of stream type (using ports) allows flexible use of the access link resources. For instance, a channel created with enough bandwidth to transport one high-quality HDTV video stream may support a variety of ports, one for a HDTV stream and others for streams that require less bandwidth. An application may activate an arbitrary set of these ports (by mapping endpoints to them) as long as there are sufficient resources available on the channel, thus sharing the access resources in real time without having to create or destroy any of the ports.

As outlined above, our goal is to support hybrid networks, in which transport is provided by heterogeneous technologies. In particular, the Touring Machine system must continue to support our current analog video and audio while expanding to support digital media transport. To this end, the link-channel-port abstractions can be used to describe analog network access: the circuit connecting desk-top equipment to the system corresponds to an access link of type "analog NTSC video;" this access link supports one channel with bandwidth equal to the total bandwidth of the channel; multiple ports of type "analog NTSC video" can be created on the channel, but only one can be active (have an endpoint mapped to it) at any given time. A useful default would be to have a system administrator create the channel and a single port of type "analog NTSC video" on it, allowing "naive" applications that do not understand links and channels to simply use the port.

2.2 Transport Topology

So far, we have discussed the new API network access abstractions for integrated digital access to the Touring Machine network. We now turn to defining the transport within the network. While how the API describes transport needs to be changed little for digital transport, supporting the new API will require substantial enhancements to the system internals for managing network bandwidth, meeting quality of service requirements, allocation of protocol converters, etc.

In the current system, *connectors* are used to define the transport topology for a session. A connector associates the *endpoints* of a session, where endpoints logically terminate media streams within the network and can be thought of as "logical ports." Connectors describe the presentation of the streams, and, in particular, define the bridging function when multiple source endpoints are included. In the next version of the system, connectors will be replaced by more general connection graphs, but the media streams emerging from a connection graph will still terminate on endpoints.

Currently, endpoints are typed by media and direction; this type must exactly match the type of the port to which the endpoint is to be assigned. (An endpoint is first *assigned* to a port, specifying to which port the endpoint will eventually be mapped, and then mapped, *activating* the port and causing the actual transport of the media stream.) In the new design, endpoints continue to be typed, but exact matching of types of endpoints and ports will not be required. Rather, endpoints and ports must be compatible, with the endpoint type allowed to be specified as a subset of the fully specified port type. Thus, for example, an endpoint of type "video" will map to any video port, whatever the detailed specification of the port.

This flexibility, transparently supported by the infrastructure, may be useful to the application programmer, who can specify transport in a uniform manner and treat all ports equivalently, even though the underlying technologies may vary widely. This flexibility allows, for instance, implicit allocation of media-format converters when the port types to which different endpoints in the same connector are mapped are compatible but not identical. In addition, by more completely specifying an endpoint type, the application programmer can restrict the ports to which the endpoint can be mapped, thus specifying the quality of service required for the media stream.

3 Related Work

Our work is related to various broadband signaling proposals which support multipoint, multimedia calls. The connector-endpoint-port abstractions of the current system for describing transport of media streams was heavily influenced by the EXPANSE signaling work [2] for broadband ISDN. Our generalizations of the abstractions to support digital transport and heterogeneous networks still has similarities with this work. However, there are significant differences, for instance in our use of endpoints and ports to type and control network access and in how the logical transport topology is specified.

A second broadband signaling proposal [3] describes a protocol that provides dynamic multipoint, multiconnection communication channels over a switched ATM network. They assume that each stream is available to all of the parties in a call, and allow users to specify which streams they receive (subject to permission policies). This proposal makes visible the underlying ATM transport infrastructure, and thus appears less applicable to heterogeneous networking.

Proposals for including non-operational interfaces into ANSA/ODP have introduced *connection networks* which register stream interfaces [4]. Transport of streams between interfaces is realized by connecting to an interface reference for a remote interface. Here, the typing of the streams is set by the connection network that supports the interface, with the connection networks organized hierarchically. Lower-level networks provide simple point-to-point transport, while higher-level networks might offer specific presentation services, such as video bridging.

Many other multimedia communications systems have used digital transport of media streams. However, this work has tended to focus on creating vertically integrated applications more than on developing a general set of abstractions useful for a broad range of multimedia communications applications and underlying network technologies.

4 Conclusion

In this paper we have concentrated on proposed changes to the Touring Machine API to support integrated digital transport of the media stream. As part of the next version of the Touring Machine system, efforts are currently underway to realize a new API that incorporates these changes, and to make its features available to the application programmer.

Other issues must be resolved before we will have a fully operational digital system. For instance, we are investigating software architectures for controlling the local terminal, including workstation-based video hardware. The proposed API allows flexible management of media streams: combined with software to control the workstation, the system will support multiple video streams from multiple communications sessions terminating on the workstation, each displayed in its own window, and each separately controllable; in addition, the port and endpoint typing mechanism offers a mechanism for automatic network-based conversion of media stream format.

We are currently working towards an initial workstation-based implementation using IP network technology. This introduces the special challenge of fitting IP-network end-to-end control of data streams into the Touring Machine control model, which up to now has tended to be more network-centric than end-to-end. More generally, we are unaware of any currently available networking technology that completely supports the abstractions we have proposed in this paper (future broadband ATM networks appear to hold the most promise). Supporting the abstractions we have proposed places requirements on network protocols; by demonstrating the usefulness to application programmers of these abstractions, we hope to influence new network protocols as they are developed.

References

1. Arango et al.: Touring Machine: a Software Platform for Distributed Multimedia Applications. Proceedings of 1993 IFIP Int'l. Conf. on Upper Layer Protocols, Architectures and Applications, Vancouver, CA, (1992); a revised version of this paper is scheduled to appear in CACM, January 1992
2. Minzer, S.: A Signaling Protocol for Complex Multimedia Services. IEEE J. Selected Areas in Comm. **9** (1991) 1383-1394
3. Gaddis, M., Bubenik, R., DeHart, J.: A Model for Multipoint Communications in Switched Networks. Proceedings of Supercomm-ICC (1992)
4. Corbett, R.: U S WEST Position Paper on Stream Interfaces. ANSA/ODP Workshop on Stream Interfaces, Boulder (Nov. 11-12, 1992)