mSTAR:
Enabling Collaborative Applications on the Internet

PETER PARNES, KÅRE SYNNES, AND DICK SCHEFSTRÖM
Luleå University of Technology

The mStar environment features an agent-based architecture, implemented in Java, which preserves compatibility with the dominant Mbone paradigm for IP multicast. In particular, mStar supports developers in creating distributed, real-time multimedia software applications such as e-meetings.

Distributed, real-time multimedia applications on the Internet permit users to cooperate in new and more interesting ways for collaborative teamwork and Net-based learning. Missing from these existing applications, however, is an integrated software toolkit that supports creating multi-user applications for real-time audio and video. Separate Mbone components do exist for this purpose, but the lack of a uniform interface can confuse users and hinder reuse.

The Java-based mStar shared software environment provides an integrated solution for generating, presenting, storing, and editing media in collaborative applications. It uses IP multicast to enable scalable distribution of real-time media and data among many simultaneous users. The powerful agent architecture that underlies the mStar environment simplifies creating new applications and encourages reuse.[1] mStar enhances both Net-based learning (distance education) and collaborative teamwork by presenting a uniform user interface for real-time audio and video, shared whiteboard, chat, voting, and distributed Web-based presentations. The system also supports on-demand recording and session playback.

Although the immediate focus of this article is the mStar environment, several distributed applications, created using mStar, are currently in use by both academia and industry. The success of those applications prompted us to found Marratech AB (http://www.marratech.com/), which offers mStar-based products for IP-multicast- and desktop-based conferencing, and pursues ongoing R&D based on the mStar environment, as well.

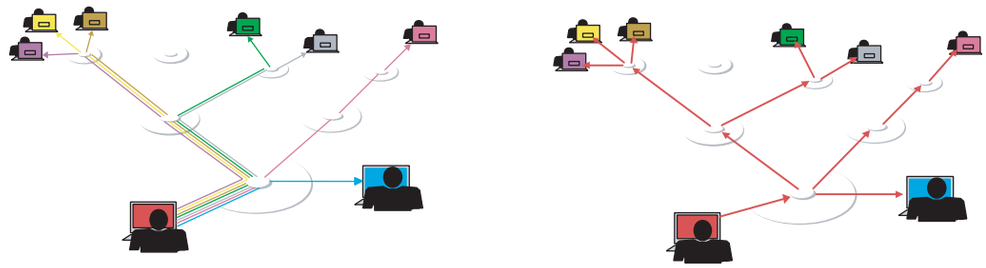
REQUIREMENTS

Scalability and decentralized control were basic design requirements for mStar. Totally distributed applications do not rely on central servers, and they have no client-server interaction. To serve these needs, we chose IP multicast for media and data distribution.

IP Multicast Communications

Traditionally, applications have used unicast to distribute multimedia data to users across the Internet. Each multimedia application either sends one identical, redundant copy to every receiver or pushes the problem into the

 1089-7801/00/$10.00 ©2000 IEEE



(a) (b)

Figure 1. (a) Unicast media distribution versus (b) multicast distribution. In (a), a single (duplicate) stream is sent from the sender to every receiver, while in (b) a single stream is sent and duplicated in the network where needed.

network using a *reflector*, which duplicates streams to every registered receiver. When the path between sender and receivers shares common network links, redundant data crosses the network. This redundancy problem is further amplified in a symmetric environment in which every session member transmits data.

As Figure 1 shows, on the other hand, IP multicast's power is in copying data only where needed in the network. One drawback to IP multicast is that it requires network router support. Service providers do not routinely enable IP multicast in routers because the router implementations are unstable, and the underlying multicast routing protocols cannot yet scale globally.

A fundamental problem with today's multicast routing protocols is that routers require an extra state for every active sender in a session. Multicast's increasing popularity, therefore, means that routers will require more memory and processing power. The IETF is addressing this problem. Deployment problems aside, however, IP multicast conserves significant network bandwidth when compared with point-to-point transmission of media data.

We addressed the deployment problem in mStar via mTunnel,[2] which lets end users of non-IP-multicast-enabled links connect to a multicast-capable domain as simply as possible. mTunnel operates at the application level and is transparent to the underlying multicast routing protocols. mTunnel permits modification to the real-time media flow depending on media type, via frame dropping or media type conversion, for example. The mTunnel encapsulation protocol also uses novel header and data compression techniques to save bandwidth—up to 15 percent on standard Mbone media flows.

IP multicast traffic uses the best-effort user datagram protocol, which can cause packet loss. This can be a problem when reliable transfer is required, such as with a whiteboard application. We solved this problem in mStar by designing the scalable reliable real-time transfer protocol (SRRTP), which is based on scalable, reliable framework principles.[3,4]

The Mbone tool suite, described in the "Related Work on Collaborative Environments" sidebar (next page), has become a de facto standard for IP-multicast media distribution. Many of the IETF's standard recommendations concerning IP multicast are based on the Mbone suite; therefore, we have built upon this work, insofar as possible, with the mStar design. Because IP multicast is still fairly new, however, the standards cover only basic, unreliable audio and video transport, session management, and session setup. Simple messaging, reliable multicast, and shared whiteboard applications are not yet standardized.

Real-Time Data Communications

The primary protocol for sending media data over the Internet with IP multicast is the real-time transfer protocol.[5] RTP functions include loss detection for quality estimation and rate adaptation, data sequencing, media synchronization, source identification, and basic membership handling. RTP operates on any kind of network and is completely self-contained. Because it does not depend on information in the network architecture's lower levels, RTP permits media traffic modification without notification to either sender or recipients. As RTP is an IETF recommendation and the de facto standard for media distribution, mStar supports it.

mSTAR AGENT ARCHITECTURE

We designed the mStar environment as an agent architecture. In mStar, an agent is a software com-

Related Work on Collaborative Environments

JETS[1] and Habanero[2] are two of the best-known Java-based collaborative environments on the Internet. In a JETS collaboration session, multiple users of any Java-enabled Web browser share, in real time, specialized applications in the form of Java applets. NCSA Habanero is a groupware application and toolkit that lets developers transform single-user applications into multi-user applications. Neither JETS nor Habanero supports real-time Internet audio and video communication, nor scalable communication using IP multicast.

The informally named Mbone tool suite is a group of IP-multicast-based media applications.[3,4] The suite includes VIC for video; VAT, RAT, and FPhone for audio; WB for whiteboard; and NTE for organized text communication. Used together, these separate tools create a "scattered" environment for distributed teamwork and Net-based learning. The advantage of creating a software environment from separate tools is that users can easily incorporate new media or tools. Disadvantages are that the separately created tools lack a uniform user interface, and their separate designs complicate reuse.

Roseman and Greenberg's[5] work on the GroupKit software environment focuses on creating distributed Tcl-based applications. Although they set out to create a complete software environment, the researchers later focused on groupware widgets, metaphors for session management, and programming abstractions for distributed applications. GroupKit is impressive but lacks support for real-time audio or video.

References

1. S. Shirmohammadi, J.C. Oliveira, and N.D. Georganas, "Applet-Based Telecollaboration: A Network-Centric Approach," *IEEE MultiMedia*, vol. 5, no. 2, Apr.-June 1998, pp. 64-73.
2. A. Chabert et al., "Java Object-Sharing in Habanero," *Comm. ACM*, vol. 41, no. 6, June 1998, pp. 69-76.
3. H. Eriksson, "Mbone: The Multicast Backbone," *Comm. ACM*, vol. 37, no. 9, Sept. 1994, pp. 54-60.
4. S. McCanne, "Scalable Multimedia Communication Using IP Multicast and Lightweight Sessions," *IEEE Internet Computing*, vol. 3, no. 2, Mar./Apr. 1999, pp. 33-45.
5. M. Roseman and S. Greenberg, "Building Real-Time Groupware with GroupKit, A Groupware Toolkit," *ACM Trans. Computer-Human Interaction*, vol. 1, no. 3, Mar. 1996, pp. 66-106.

ponent that resides within an application and is responsible for specific tasks. An agent can have a graphical interface, although that is not a requirement. Figure 2 shows agents running within Maratech Pro, a desktop-based, e-meeting application that lets users interact in real time using different media. Media examples include real-time live audio and video, shared Web pages, distributed whiteboard, and chat. The application also allows for generic recording and playback of real-time media.

We used an agent architecture because it simplifies the reuse of self-contained parts in other applications, such as the important network agent (common to all network-aware mStar-based applications) or the simpler chat agent. The architecture also lets agents communicate with and directly control explicit parts of applications.[4]

mStar also features a special group of agents, called *mediators*. These agents translate between internal messages (formatted in a common control bus language that all mStar agents use to communicate with each other, as we explain later) and external messages specific to the application or module to which the agent interfaces. For example, the Web agent controls external browsers while internally providing a uniform interface. This allows other agents within an application to control external Web browsers without having to know which browser it is or how it is controlled.

Inter-Agent Message Communications

Agents communicate in mStar via the control bus, which is designed for simple, self-contained messaging. Self-contained messages contain all needed information during agent exchanges; messages should not depend on underlying data referencing nor on a predefined interface, which is the case with, for example, JavaBeans and CORBA. mStar agents use the control bus to transmit messages both within applications and between different instances of applications. The control bus can be used on any underlying reliable transport protocol. Also, agent developers can choose to use either the mStar control bus or a protocol of their own, as we did with the mStar whiteboard application.[4]

Agent and Network Communication

The mStar environment allows developers to readily create new connections between the network and an application agent. Although the underlying network protocols are not connection oriented, we use the term connection as a programming metaphor inside the environment.

After creation, the connection is used to exchange data and control information with the network. Data received from the network is stored internally within the application in its original form with RTP headers, but parsed into a more usable data structure. In keeping with the Application-Level Framing paradigm, network protocol details are not hidden from the receiving agent.[6] With ALF, data handlers are involved in the network process because they can best decide how to handle data affected by packet loss or network jitter.

Application agents open a network connection by registering with the network agent and providing an appropriate network address and port. Network data is queued for later handling, which allows agents in multithreaded applications to handle incoming data at their own pace while letting the current network thread continue receiving data.

When the agent opens the connection, it also requests a quality-of-service level. Developers can specify reliable or unreliable transport; a third level—differentiated quality of service via RSVP[7]—is currently being added to the environment in collaboration with the authors. With unreliable transport, data is packaged into RTP packets and sent over the physical network. For reliable transport, the mStar network agent can use any reliable transport mechanism available. The latter means that the application agent developer should not take for granted which protocol will actually be used at runtime. The developer should instead see the connection as a reliable transport mechanism where the data will be delivered to a number of receivers.

The Internet protocol provides only that data being sent will arrive at its destination. The data might be out of order, due to packet reordering or retransmission, but out-of-order problems are not resolved at the network level. The handling agent must resolve them, if possible or necessary. Agents that must receive in-order data can use a software utility that filters the queue.

mStar agents divide their outgoing data into appropriate data chunks to preserve the ALF paradigm. The network agent adds the correct data header—typically a 12-byte RTP header—to outgoing packets. To minimize the number of copy operations at the application level, the agent provides the network with a data buffer that can accommodate the appropriate protocol header. If this extra space is not available, mStar allocates a new buffer that leads to a performance penalty, which we describe later.

This division between applications and network access code functionalities enables fast deployment of network code modifications without modification to other applications using mStar. The separation also permits the underlying IP transport mechanism to change transparently between unicast and multicast, for instance, as needed. Moreover, with unicast, separation allows data to be transported reliably via TCP instead of SRRTP; SRRTP is optimized for reliable transport between a group of members, whereas TCP is better optimized for point-to-point traffic.

As IP multicast is not available everywhere, it is important that end-user applications dynamically adapt, via a reflector, between unicast and the more scalable—and preferable, to save bandwidth—multicast. By introducing a separation between application agents and the network agent, as done with mStar, an application can dynamically switch

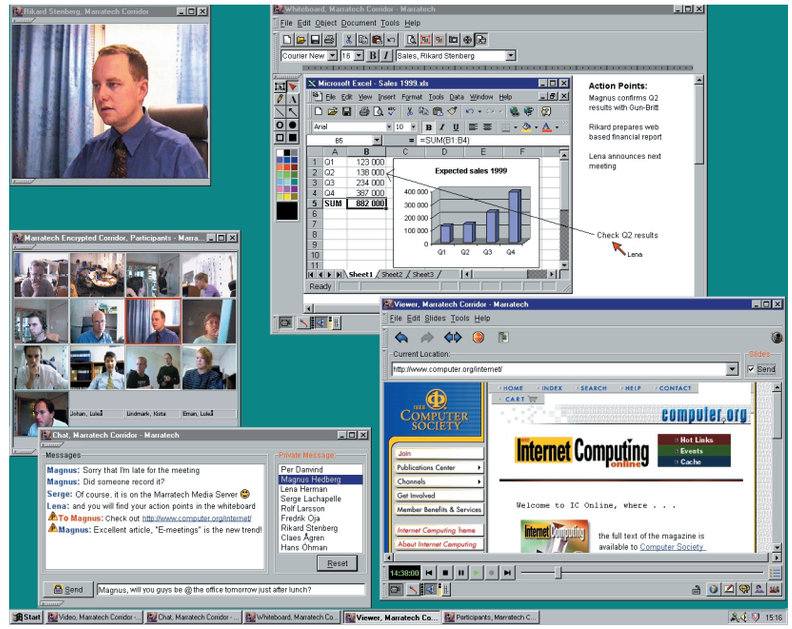


Figure 2. A screenshot example of Marratech Pro, an e-meetings application built with the mStar environment. The participants can be geographically dispersed. "Meetings" are arranged in several ways: Sessions can be broadcast on a predefined multicast address; a session description file can be uploaded to a Web server or e-mailed to expected participants; or participants can be invited via a session initiation protocol. For more details, see http://www.marratech.com.

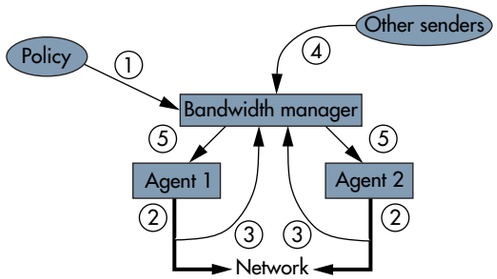


Figure 3. The bandwidth management process: (1) On initialization, the bandwidth manager fetches bandwidth policy information. (2) Agents start to transmit. (3) The bandwidth manager monitors the amount of data agents are transmitting. (4) At the same time, the manager monitors the bandwidth used by other senders in the session. (5) If necessary, the bandwidth manager notifies the agents to modify their notion of allowed bandwidth.

between these data distribution paradigms. This in turn makes the application more adaptable to existing network conditions.

BANDWIDTH MANAGEMENT

Bandwidth management is essential, though not easy to achieve, in a distributed-multimedia software environment as there are many simultaneously active media senders. Distributed-media applications must adapt to different network environments and different amounts of available bandwidth. Many existing IP-multicast-based applications base their bandwidth usage on the network scope; for example, media streams aimed at a local organization can use more bandwidth than a media stream transmitted to an entire continent.

Designers of a reusable bandwidth management mechanism must balance code reusability and application control. The more the mechanism is tied into the application, the better bandwidth control it maintains, although it complicates reuse in new applications.

The mStar environment balances bandwidth management with a *bandwidth manager agent*, which controls agents and determines how available bandwidth should be utilized.

As Figure 3 shows, the bandwidth manager fetches policy information at initialization. This information can be predefined for a specific application and media, or it can be gathered depending on information in the session description file[8] for the current session. The bandwidth manager monitors the data being sent and the bandwidth used, and notifies agents if bandwidth allocations must change.

Notifying agents of different bandwidth allocations might result in different behaviors from agent to agent. For example, if the new allowed bandwidth is less than the amount requested by a user, the video sender agent would set its target bandwidth to the allowed bandwidth. Alternatively, if the allowed bandwidth is greater than the amount requested, the agent sets the target bandwidth to the requested level. To avoid agent starvation, the value of the allowed bandwidth is never accepted by the bandwidth management agent below a certain threshold. These steps are repeated regularly to let the application adapt to the session.

As Figure 4 shows, the bandwidth policy might span numerous scenarios in a session:

- Case A: one media, local user (local scope). The bandwidth manager fulfills the user-requested static bandwidth.
- Case B: one media, all users (session scope). The bandwidth manager maintains constant media bandwidth but adapts local target bandwidth to serve other senders and tries to maintain a constant bandwidth usage for that specific media.
- Case C: all media, local user. The bandwidth manager sets the total available bandwidth to span all media for the local user. The resulting policy keeps the local total target bandwidth constant over all media (that is, bandwidth information about other senders will be ignored).
- Case D: all media, all users. Although basically the same as scenario C, the bandwidth manager here adapts the target bandwidth to satisfy all senders in the session.

The bandwidth policy information can be further controlled remotely via the mManager framework.[4]

mStar's bandwidth management architecture enables agents to monitor many other parameters besides utilized bandwidth, including reported packet loss and propagation jitter. In our current implementation, however, this information is not used to calculate the target bandwidth.

MOBILE-DEVICE SUPPORT

Increasingly, collaboration today takes place away from the office, which accounts for the growing popularity of mobile devices—pocket PCs or palmtop PCs like 3Com's PalmPilot. The mStar environment, via its runtime library, enables developers to transparently move client applications to the limited platform of mobile devices. The devices' network access capacity is of particular concern

because, although some handheld devices support high-speed connection such as Ethernet or high-speed wireless LAN, the devices' CPU power limits them from sending and receiving at full speed.

Scalability requires that software applications adapt to different runtime platforms and operating systems, and to varying resources: memory, computing power, screen size, and screen color depth, for example. These requirements are more acute with handheld devices because the devices' limited capabilities require modifications to real-time data before the receiving devices can display it.

Figure 5 shows our solution, in which a media gateway (proxy) transcodes—modifies—the data as close to the final recipient as possible. The transcodings take place on either the network level or the media level, depending on application type.

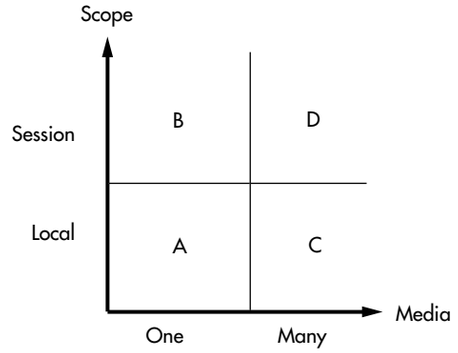


Figure 4. Different policy scenarios. Case A: one media, local user; Case B: one media, all users; Case C: all media, local user; and Case D: all media, all users.

Network Level

At this level, a media gateway built using mStar modifies the traffic flow independently of the media being transported. The media gateway also acts as a gateway between multicast and unicast. Multicast is typically not supported by handheld devices nor the dial-up connection they use to access the Internet.

Most real-time applications handle packet loss, which typically results from unreliable UDP data, in media flows. Similarly, a sending device drops packets if network congestion occurs, as the sender does not immediately have to adapt its transmission rate. High-speed flows can be transported over low-bandwidth links without involving the original sender.

A special problem occurs with the TCP/IP stack on the PalmPilot, which handles TCP-only traffic, with built-in data retransmission. This situation puts extra constraints on the transcoder to transmit exactly the right amount of data to the handheld device. Many applications require a pull-architecture, in which devices fetch data on an as-needed basis. The amount of transmitted data also depends on the device's user interface: Unlike PCs, the limited screen resolution typically prevents the simultaneous display of different media interfaces.

The overhead of using IP, UDP, and RTP is significant, so mStar resolves both RTP and real-time transfer control protocol (RTCP) issues via the control bus. More typically, however, developers handle specific RTP and RTCP issues in the gateway, not in the handheld device. As the agent metaphor lets agents communicate independently of the application they are running in, the media gateway architecture can be used both within, and between, applications.

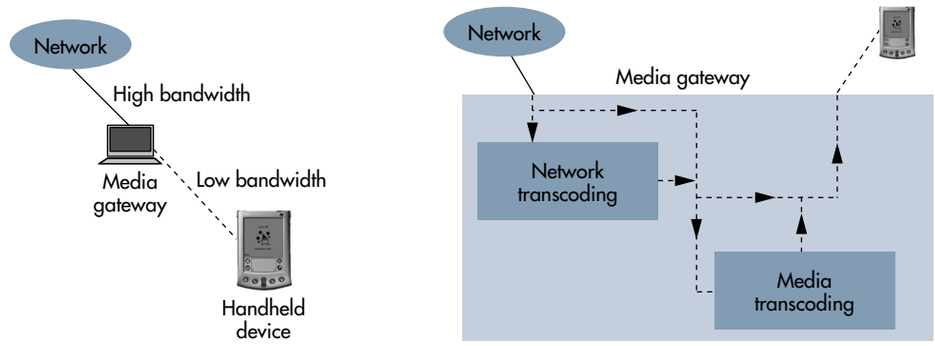
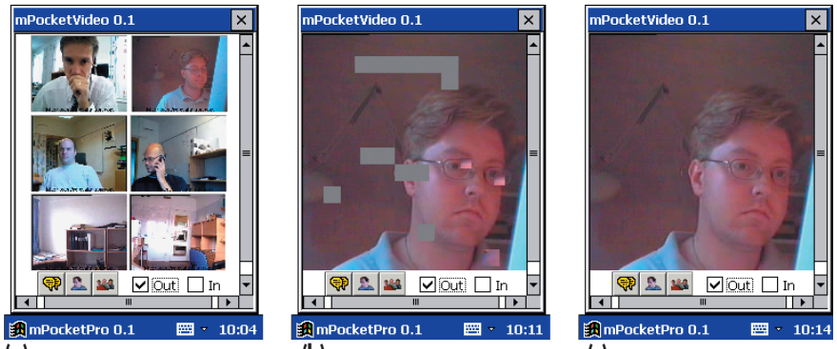


Figure 5. The media gateway: (a) media scaling to accommodate a small handheld device; (b) internal architecture of the media gateway.



(a) (b) (c)

Figure 6. The mPocketPro prototype user interface. (a) Corridor mode showing several active video streams; (b) full video mode showing one video stream with bandwidth adaptation turned off; and (c) full video mode with bandwidth adaptation turned on.

Media Level

Because real-time media, such as video, cannot be displayed on handheld devices in original form, data transcoding is performed at the media level, a step above the network level. Real-time video is typically converted into lower resolution (grayscale) or lower color depth. With enough CPU power available, the device could make the conversion; typically, however, a media transcoder does the decoding and re-encoding of the video flow for device display. Other real-time media types are handled similarly.

EXAMPLE: MOBILE E-MEETINGS

The mStar environment is ideally suited for applications, notably e-meetings, that can be conducted with handheld devices such as the mPocketPro prototype that we are developing. mPocketPro permits real-time-audio, full-duplex conferencing and renders several live video streams coded with ITU H.261, which is the most common video format for Mbone applications. The underlying hardware is a Casio Cassiopeia E-115, which has a color screen resolution of 240 × 320 pixels in 16-bit color. Figure 6 shows the user interface.

Access Modes

The mPocketPro runs in both direct network and media gateway access modes. In direct network mode, the device connects directly to the Internet, receiving and transmitting media streams directly. The application receives all data but filters out parts of the stream that it cannot handle. For example, the mPocketPro receives and displays a maximum of 100 to 150 Kbps of video before it depletes its CPU resources and starts dropping parts of the video stream. In contrast, a 700-MHz Pentium III running Windows 2000 can easily display several Mbps of video. An application that receives too much video will display block artifacts, as Figures 6b and 6c show.

The video codec design lets mPocketPro display video, which allows for parts of the stream to be decoded despite the whole stream's not being received. Most existing codecs lack this quality. If mPocketPro receives too much video, of course, its ability to handle other real-time media is diminished; that is, the more important media—audio—will suffer quality loss.

In gateway access mode, mPocketPro receives modified media streams transcoded by a media gateway, instead of receiving media streams directly as Figure 5a showed. The multicast GateWay (mGW), which runs on a more powerful server computer, scales the media streams to the bandwidth requested by the mPocketPro application. Scaling is done by recoding the active streams into a lower bandwidth and/or a lower frame rate. The mGW decodes all active video streams (just as the mPocketPro would do in direct network access mode) and re-encodes each stream. The disadvantage, of course, is the optionally reduced image quality and the lowered frame rate of the displayed video.

Benefits

The mPocketPro and the mGW designs are based on the mStar agent architecture and so communicate with each other using the common control bus. Because the device and the gateway need not know where they run in the network or in which application they operate, the mPocketPro can be readily embedded into a larger application without changes to its gateway communication.

An additional benefit of media gateway access mode is that the client can request that the server recode active media streams differently depending on the user. For instance, when viewing a full-screen video image of one active video stream, the user probably wants to receive that stream in higher bandwidth (and thus higher quality) instead of bandwidth evenly allocated to each active stream. The client (mPocketPro) will thus request the gateway to change its encoding allocation priorities accordingly.

The recoding also lets the mPocketPro receive video streams in other codes that can then be decod-

ed and re-encoded into H.261. To test this behavior, we integrated an MPEG4 decoder (using the DivX codec) into the gateway. To handle video streams with resolutions too high to display on the mPocketPro, we let users scroll the video images on-screen in pan-scan mode. As users scroll, commands are sent to the gateway, which then "scrolls" its encoder accordingly, sending only the displayed part of the video to the mPocketPro. This "remote scrolling" allows the end user to select parts of a video stream to view.

FUTURE WORK

Work with the distributed-media mStar environment continues, in both academia and industry. The current research focus is on extending mStar with generic functionality that is essential for a complete media software environment. For example, researchers are investigating how to provide a more robust and stable IP-multicast environment, known as ubiquitous multicast access. In this environment, the underlying network software should adapt to either unicast or multicast, transparently to users.

Another research area involves the original sender of the media streams (primarily video) in the scaling process for adapting to mobile devices. Researchers are determining how to require the original sender to transmit the media data divided into several connected streams (layers), which lets the receiver choose how many layers to receive depending on available bandwidth.

Investigators are also evaluating a generic, semantic-reliable multicast protocol for use by applications that operate on distributed data, which can be divided into semantic building blocks. Finally, digital video broadcast combined with Internet conferencing is yet another research topic. ■

ACKNOWLEDGMENTS

We thank all those who reviewed this article, especially the anonymous reviewers, as well as Louise Burnham, who edited it. We credit all the hard workers at Marratech AB for bringing the mStar ideas to fruition. This work was done within Esprit projects 20598 Mates and 28410 Roxy, which are supported by the European Union's Fourth Framework Program, Information Technology section, and the SIRAM project, supported by the Swedish Research Institute for Information Technology. The Centre for Distance-Spanning Technology (CDT) and Marratech AB also provided support.

REFERENCES

1. P. Parnes, *The mStar Environment—Scalable Distributed Teamwork Using IP Multicast*, licentiate of engineering thesis, Dept. of Computer Science and Electrical Eng., Luleå Univ. of Technology, Sweden, 1997.
2. P. Parnes, K. Synnes, and D. Schefström, "Lightweight Application-Level Multicast Tunneling Using mTunnel," *Computer Comm.*, vol. 21, no. 15, 1998, pp. 1,295-1,301.
3. S. Floyd et al., "A Reliable Multicast Framework for Lightweight Sessions and Applications Level Framing," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, Dec. 1997, pp. 784-803.
4. P. Parnes, K. Synnes, and D. Schefström, "Real-Time Control and Management of Distributed Applications Using IP-Multicast," *Proc. Sixth IFIP/IEEE Int'l Symp. Integrated Network Management (IM 99)*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999.
5. H. Schulzrinne et al., "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 1889, Jan. 1996; available online at http://www.ietf.org/rfc/rfc1889.txt.
6. D. Clark and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *Proc. SigComm*, ACM Press, New York, 1990, pp. 201-208.
7. R. Braden et al., "Resource ReSerVation Protocol (RSVP)," IETF RFC 2205, Sept. 1997; available online at http://www.ietf.org/rfc/rfc2205.txt.
8. M. Handley and V. Jacobson, "SDP: Session Description Protocol," IETF RFC 2327, Apr. 1998; available online at http://www.ietf.org/rfc/rfc2327.txt.

Peter Parnes is an assistant professor at the Centre for Distance-Spanning Technology at Luleå University of Technology, Sweden. His research focuses on scalable distributed applications based on IP multicasting and how such tools can be used to improve daily work, electronic meetings, and distance education. Parnes has a PhD in computer science. He is also a cofounder of the Marratech AB company where he works as a software architect and is a board member.

Kåre Synnes is a graduate student in the Department of Computer Science at Luleå University of Technology, Sweden. His research interests include IP-multicast audio tools, distributed platforms for cooperative work, and Net-based learning. Synnes is a cofounder of the Marratech AB company.

Dick Schefström is R&D director at the Centre for Distance-Spanning Technology, Luleå, Sweden, and has 15 years' experience with industrial software development and telecommunication applications. Schefström earned a PhD in computer science. He is a cofounder of the Marratech AB company and has written more than 20 scientific papers, has edited a book on software development environments, and has been extensively involved with several joint European R&D efforts.

Readers can contact the authors at Luleå University of Technology, CDT, 971 87 Luleå, Sweden; {Peter.Parnes, Dick.Schefstrom, Kare.Synnes}@cdt.luth.se.