# User Empowered Virtual Multicast for Multimedia Communication

Eva Hladká
Laboratory of Advanced Network Technologies
Faculty of Informatics,
Masaryk University Brno,
Botanická 68a, Brno 602 00, Czech Republic
Email: eva@fi.muni.cz Phone: +420 549 493 535

Petr Holub
Faculty of Informatics and
Institute of Computer Science
Masaryk University Brno,
Botanická 68a, Brno 602 00, Czech Republic
Email: hopet@ics.muni.cz Phone: +420 549 493 944

Jiří Denemark
Labor. of Advanced Network Technologies
Faculty of Informatics,
Masaryk University Brno,
Botanická 68a, Brno 602 00, Czech Republic
Email: jirka@ics.muni.cz

*Abstract*— **We introduce concept of user empowered UDP packet reflectors to create virtual multicasting environment as an overlay on top of current unicast networks. The virtual multicast is used as a bottom layer for secure and efficient collaborative environments. The end-users' ability to fully control this environment—in a way similar to peer to peer networks—is the primary advantage of our approach. Other interesting features that are possible only in virtual multicast environment are also discussed in this paper.**

*Index Terms*— **virtual multicast, UDP packet reflector, user empowered approach, modular architecture**

## I. Introduction

Multicast is the "natural" solution for a group communication [1]. Multicast communication can be characterized by the following statement: "The same data are transferred at most once on any particular link". This implies large ("infinity") scalability, but imposes non-trivial requirements on the network as all the network nodes must support it in a consistent way. Despite continuous effort only very small fraction of places on Internet have reliable native multicast connectivity. While the radio, television, and other mostly one-way broadcasting systems are practically impossible to be deployed on large scale without native multicast support, the collaborative environment usually connects bidirectionally few places only and "infinity" scalability is not such pressing issue. The communicating groups have usually at most 20 sites connected while larger groups need very precise orchestration and moderation. All the practically used multicast protocols have also other disadvantages: it is near to impossible to take care of quality of service requirements for the whole multicast group, it is very difficult to provide secured environment without a shared key, and there is no easy support for accounting.

This research is supported by a research intent "High Speed Research Network and its New Applications" (MSM000000001).

These problems may be overcome through multicast connectivity simulation, where active nodes have a role of *reflectors* ("mirrors"), that replicate all traffic passing through them in a controlled way. In such environment multicast videoconferencing clients can be used with ease while keeping the advantages of unicast point-to-point communication lines—thus creating *virtual multicast environment* (this technology is used e. g. in VRVS [2] or AccessGrid [3]). The reflectors can even transform the incoming traffic and can be directly controlled by the end users. These mirrors play a role of multicast join-points, allowing clients to connect (and drop out) without any undesired influence on the rest of the group.

We propose reflector architecture based on active router architecture [4]. This architecture can be used for creation of *ad hoc* overlay networks, where both mirrors and the overlay network creation is administered directly by end users. The behavior of each individual mirror can be independently controlled, including the security environment. The security context may be individualized for each client, using any authentication scheme, including PKI, Kerberos or shared keys. While reflector technology is only partially scalable it is the most efficient infrastructure for groups with no more than tens of clients. The main advantage is flexibility, user empowered-ness, and independence on any specific network features except for simple unicast routing. All the "advanced features" are provided by higher, user controlled layer. Any user group can start its own mirror and only unicast connectivity is required from any client to the mirror. Two or more mirrors may be combined to provide a true overlay network. While data routing and replicating are the basic functions, many more services can be provided within this framework. Varying demands of different users' groups and even specific demands of individual users within a group can be handled by specific extensions (modules) to the basic mirror program in the active network framework. Few examples of already implemented features are: full logging and data recording, data encryption and decryption, synchronization of streams, authentication, authorization and accounting, and stream traffic shaping.

## II. Reflector architecture

The design of a reflector must be flexible enough to allow implementation of required features and leaving space for easy extensions for new features. This leads to a design that is very similar to our active router architecture [4] modified to work entirely within the user space. Users without administrator privileges are thus able to run reflector on any machine they have access to. The reflector architecture is shown in Fig. 1.

### A. *Data routing and processing*

Data routing and processing part of the reflector comprises *network listeners*, *shared memory*, *packet classifier*, *processor scheduler*, number of *processors*, and *packet scheduler/sender*.

Network listeners are bound to one UDP port each. When packet arrives to the listener it places the packet into shared memory and adds reference to a *to-be-processed queue*. The packet classifier then reads packets from that queue and determines a path of the data through the processor modules. It also checks with routing AAA module whether packet is allowed or not (in the later case it simply drops that packet and creates event that can be possibly logged). Zero-copy processing is used in all simple processors (packet filters), minimizing processing overhead (and thus packet delay). Only the most complex modules may require processing that is impossible without use of packet copies.

The *session management* module follows the processors and fills the distribution list of the target addresses. The filling step can be omitted if data passed through a special processor that filled the distribution list structure and marked data attribute appropriately (this allows client-specific processing). Processor can also poll session management module to obtain up to date list of clients for specified session. Session management module also takes care of adding new clients to the session as well as removing inactive (stale) clients. When new client sends packets for the first time, session management module adds client to the distribution list (data from forbidden client has already been dropped by packet classifier). Information about the last activity of a client is also maintained by the session module and is used for pruning stale clients periodically. Even when distribution list is not filled by the session management module, packets must pass through it to allow addition of new clients and removal of stale ones.

When the packet targets are determined by the router processor a reference to the packet is put into the *to-be-sent queue*. Then the packet scheduler/sender picks up packets from that queue, schedules them for transmission, and finally sends them to the network. Per client packet scheduling can also be used for e. g. client specific traffic shaping.

The *processor scheduler* is not only responsible for the processors scheduling but it also takes care of start-up and (possibly forced) shutdown of processors which can be controlled via administrative interface of the reflector. It checks resource limits with routing AAA module while scheduling and provides back some statistics for accounting purposes.

### B. *Administrative part of the reflector*

Communication with the reflector from the administrative point of view is provided using *messaging interfaces*, *management module*, and *administrative AAA module* of the reflector. Commands for the management module are written in a specific *message language*.

Messaging interface is generic entity, which can be instantiated as e. g. RPC, SOAP over HTTP, plain HTTP interface with SSL/TLS or GSI support, or simple TCP connection bound to loop-back interface of the machine running the reflector. Each of these interfaces unwraps the message if necessary and passes it to the management module.

Management module checks validity of the message and its authenticity and authorization status, querying the administrative AAA module, which is also responsible for the processing of accounting information of the accepted messages. Availability of sufficient resources to process accepted message is checked with resource management module and the message is passed to the appropriate module(s) afterwards. Completion status is returned back to the management module, which notifies the administrative AAA module to close the accounting. If a failure occurs, its description is stored via the administrative AAA module, an error that can be logged is generated, and an error message is simultaneously sent back the client via messaging interface the client is connected to.

The same mechanism can be used for *logging* purposes. One or more messaging interfaces may be opened with the `LOGGING` flag set and such messaging interfaces receive all events created within the reflector via the management module. The way how to receive logging information through some messaging interface, which doesn't have the `LOGGING` flag set, is to send a request asking for logging information via this interface.

A message language for communication with the management module is called Reflector Administration Protocol (RAP) [5]. It is a text-based request/response protocol that uses US-ASCII character set. Lines are delimited by character pair CR and LF (0x13 and 0x10). Protocol message can be either user's request to the reflector or response of the reflector to the user. One request can be followed by multiple responses. Protocol is designed as soft-state, i. e. connection is closed after certain period of client inactivity. Keep-alive messages have to be sent if the client wants to maintain connection and has no requests to send.

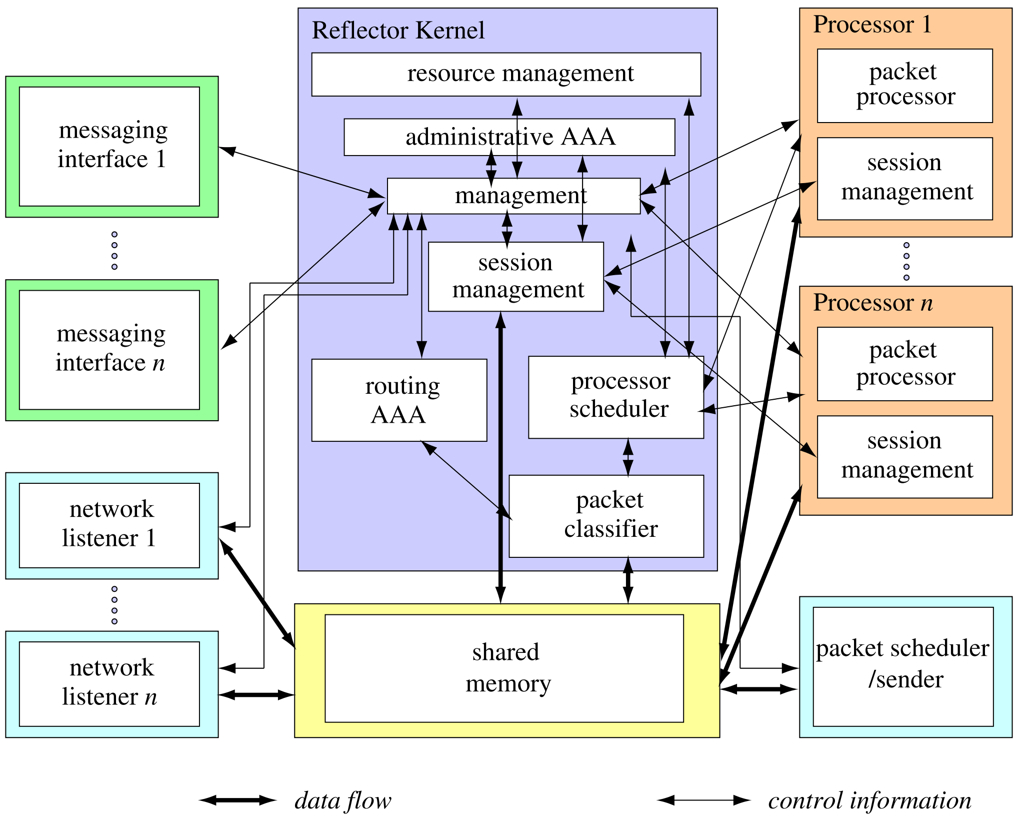

Fig. 1. Reflector Architecture

Each message comprises message headers and message body. In case of request, the headers contain

- specification of method (that is an actual command for the reflector) and method specific headers
- target module reference (RAP contains a module addressing schema enabling to pass requests to specific modules),
- protocol version identification, length of the message body, possible request to process the request in blocking manner, and other auxiliary information

Message body contains method specific information if needed. Methods (or commands) can be divided into two groups: general reflector methods and extending module specific methods. Group of general methods covers methods for requesting various information on status of the reflector and its modules (either in interactive way or in form of subscription for logging information), commands for manipulating both routing and administrative access control lists as well as administrative user database, and commands for controlling modules (starting, stopping, and restarting modules).

There are several response classes used by reflector to reply to client's request: 100 (for informational and logging messages), 200 (for successful completion of user request with message body containing actual response data), 400 (for client request errors), and 500 (for server-side error).

Complete definition of RAP version 1.0 including formal definition using ABNF [6], detailed protocol description, and example communication using RAP can be found in [5].

## III. Advanced Reflector features

The basic function of the reflector is retransmission of received data to one or more listeners. This can be easily extended to support other useful functions. The reflector replicates all the traffic coming through specified port to all the clients connected to that port. Clients do not interact in advance—they just connect to the reflector to automatically receive all the traffic sent to the reflector and also all the client traffic is automatically distributed by the reflector. The reflector security (per port or per client) policy may change this behavior and forbid some clients from listening or sending data.

*a) Tunneling and scalability:* The scalability of the reflector based communication can be increased using tunneling between the reflectors. Possible tunnel configurations are:

- *full-mesh tunneling* – Each reflector has addresses of its peers and sends data received from directly connected clients to all the peer reflectors and accepts data from all the peer reflectors and distributes it to all directly connected clients. This is the least efficient way with respect to scalability but has the simplest setup.
- *static tunneling* – Routing among reflectors is done by manually pre-configured static way. Such tunneling is

used for MultiSession Bridge in AccessGrid[7]. This can be efficient for long-lived network of reflectors with infrequent changes. It is not suitable for networks created in *ad hoc* mode. When properly managed, this is the most efficient configuration.

- *dynamic tunneling and routing* – Mimics the behavior of routers and may use very specific routing algorithms (even multi-criteria). The simpler configurations may use distance vector routing algorithms used in `mrouted` (on the MBone) or even algorithms from peer-to-peer networks might be used (e. g. an algorithm developed in Pastry project [8]). This option is the most suitable for *ad hoc* networks of reflectors and may be the most efficient for dynamic environments.

Both static and dynamic configurations are suitable for networks with some low bandwidth or high latency lines since they can be configured to bypass such links.

*b) Logging:* Following events are examples of events generated within the reflector either as a result of data transmission or reacting to managerial decisions: start-up and shutdown of the reflector, beginning and end of data transmission, start and stop of data recording, users' login and log-off times and login failures, non-authenticated requests to join a group, program errors. These events are sent to all the messaging interfaces subscribed for logging information, which store them into disk files or pass them on e. g. the system `syslog` interface or to some external monitoring tools.

*c) Stream transcoding:* A specific module can act as a multimedia stream transcoder. This is usually used when some client is connected via the low bandwidth link. As the transcoding consumes rather lot of processing power, a specific reflector—the *gateway*—may be set up at the beginning of the low bandwidth link and connected to the nearest reflector using the tunneling capability and thus enables clients connected via low bandwidth links to participate without influencing clients connected using fast links.

Another situation in which the transcoding feature is useful is when media stream is produced and transferred in format that is not acceptable for the some client(s).

*d) Video stream composition:* In some circumstances, like different quality of links used by participants, large groups resulting in too many windows at client sites, insufficient processing power at client sites to decode large number of simultaneous streams etc., it may be advantageous to down-sample video streams and compose several of them into one stream directly on the reflector (provided it has sufficient memory and processing power). The first version of such a processor is described in [9] and it has already been adapted to fit into the new architecture. This processor is based on the `vic` tool and support exactly the same set of video formats. Up to four video streams can be composed into one output stream. Input video formats are auto-detected, the processor is able to work with different formats simultaneously. The output video format is configurable by the end user.

*e) Recording:* The centralized data recording facility has many advantages over much simpler features of most videoconferencing clients. It may be a trusted neutral agent (no local editing of content), it can guarantee to store all data actually transmitted (in contrast to local client which may experience some local data loss), it may be more efficient (just one copy of the data is created), and it provides recording capabilities independent of whether the client software has recording capabilities or not. The recording processor within the reflector is controlled by end-users, stored data can be easily used to re-play the communication. Storage of all the data transmitted allows a synchronized re-play of not only audio or video but also of shared workspace modifications, too.

*f) Synchronization:* Several independent UDP streams may be synchronized using the appropriate processor. It inspects timestamps and delays or reorders packets within a specified time windows so the resulting outbound streams are fully synchronized. It uses timestamps in RTP packets and is able to work with independent time references ("zero time") and increments for each stream (even different streams from the same source computer can use different time references [10], [11]). The information needed for converting between real (absolute) time and relative RTP time is taken from RTCP packets. This requires sending computers to have their real time clocks synchronized e. g. using NTP protocol.

This synchronization feature can be used e. g. to send 3D video in two streams for each eye separately synchronously over best effort network. Such transmission allows to use more demanding video processing compared to sending it in one stream since the processing can be distributed among two or even more machines provided they have their real time synchronized properly.

A preliminary implementation, which uses dedicated reflector lacking our new modular architecture, has been described in [12].

*g) Traffic Shaping:* The traffic shaping processor supports among other: bandwidth limiting, delaying, deliberate packet loss, and packet duplication (on the same stream). The last two features are used usually for debugging purposes or for simulation non-ideal network conditions. Delaying is used to increase fairness among videoconferencing partners in unfair conditions (where one or more partners have substantially larger delays).

*h) Raw Data Dumping:* While events from management module are stored via the logging interfaces, the reflector supports also raw data dumps of all incoming packets (including those rejected for authentication or authorization reasons). The reflector, running in the user space, does not rely on the `tcpdump` (requiring root privileges) and in the dumping mode it simply copies all data on inbound interfaces into a file for later analysis.

## IV. Security in context of the reflector

The reflector is a program started by an ordinary user—who thus becomes primary reflector administrator—and it runs under his or her identity. As this user grants some privileges to partners within the group, the reflector must protect user from malicious behavior of third parties. This is done via authentication and authorization mechanisms that are part of the administrative AAA module. In various scenarios (e. g. military or bioinformatics) the actual data communicated among partners must be protected as well.

All the administration is done via secure messaging channels (e. g. SSL/TLS secured HTTP). User can authenticate using login and password or via some authentication credential (e. g. Kerberos ticket or X.509 certificate). The authorization is done using ACL (access control list) and is performed per command (similar to the TACACS authorization mechanism). The reflector administrator creates the first ACLs and also specifies (during compile-time and reflector startup) which authentication mechanisms will be supported.

*i) Basic end-user security:* Simple client authorization is based on IP address restrictions. Appropriate "accept" and "deny" ACL records contain IP addresses or subnets (defined as an IP addresses with associated netmasks). The decision is taken by the routing AAA module, rejected packets are dropped and appropriate event is generated. This decision precedes the session management phase to eliminate work that would be otherwise discarded. However, session management must be informed about changes in ACLs to be able to discard forbidden clients immediately.

Restrictions based on user names are done indirectly—a user connects via secure messaging channel and adds his/her IP address into the list of accepted IP addresses. In such scenario it is also possible to employ soft-state mechanism when certain IP address or address range is accepted over limited period of time only. The user is responsible to renew the authorization in regular intervals.

*j) Strong end-user security:* Areas like military and medicine require strong security support. This issue was studied in [13] and there is a solution currently available for the reflector.

The secure reflector consists of four parts: initialization, authentication, communication, client disconnection. The *initialization phase* processes the input parameters, initializes cryptography subsystem and waits for client to connect.

In *authentication phase* client and server set up secure channel using RSA secured TCP connection. The client authentication is then based on user login and password. When accepted, the secure connection is maintained during the whole conference as it is used for session tear-down and optionally for redistribution of keys if temporal re-keying is enabled.

In *communication phase* reflector forwards data encrypted using symmetric AES cipher using key exchanged during authentication phase with clients.

In the final *client disconnection phase* client asks for end of session and is removed from the list of allowed clients by the server. Disconnection phase can also be initiated by the server when it shuts down.

Client side is realized by specialized local reflector which client tools (e. g. `vic` or `rat`) connect to[1]. This local reflector processes the authentication, exchanges session keys with the reflector (each client/server pair has its own session key) and is responsible for data encryption and decryption using these session keys.

*k) Use of reflectors in adverse networking environments:* *Firewalls* are spread in many places and are the administrative solution to protect LANs and their resources from malicious users and accidents. This protection has a negative side effect since it means barrier to free network communication and makes difficulties when deploying applications relying on user empowered paradigm in case when the firewall is not controlled by the end user. It is usually difficult to achieve reconfiguration of firewall for communication on unusual ports. Another problem that often goes hand in hand with firewalls is *network address translation (NAT).* It usually doesn't work for TCP connection initiated from outside of inner network and for UDP traffic directed inside, the latter of which is a serious problem for virtual collaborative environments that rely on bidirectional UDP traffic for multimedia content.

A possible solution for reflector based communication may work without touching firewall configuration at all and it may also solve problems with NAT. First, we need to use two reflectors—one inside the zone protected by firewall and/or NAT and the second one outside. Reflectors need to have special processor that performs encapsulation of packets to pretend that they belong to some well known protocol that is passed through by most of the firewalls (HTTP being a good example). Our experiences show that this scenario can be deployed quite successfully[2] [14]. Such scenario works fine unless both clients are hidden in different networks performing NAT. In such case they need some reflector on outer public network that forms a kind of rendezvous point toward which both communicating clients point the tunnels of their local reflectors. Otherwise it may be impossible to address each other directly.

The client reflector is the one, which initiates the TCP connection with the desired address and port number of main reflector because of possible NAT, while the main reflector outside of the firewall waits and accepts the incoming TCP connections. RTP packets are sent through the TCP connection with minimal additional header prepended. The header consists of two 4 byte numbers, one carries the RTP packet length and the other is a flag distinguishing RTP data packets from RTCP packets.

## V. Future work

The work we are targeting now is to implement all described features to the reflector. For the future we plan to

[1]Some of client MBone tools require a small modification to be able to connect to the reflector running locally as they bind on the same port they are connecting to obviously resulting in conflict in such setup.

[2]The packet reflector modified for communication through firewall was tested successfully with voice communication in challenging environment (two wireless LAN hops and application layer firewall) between ICN'01 conference in Colmar, France, and Masaryk University in Brno, Czech Republic.

continue to support pilot user groups to get stimulating reactions and new ideas as this reflector has always been developed for group communication of real groups of users and lots of features are based on their demands. The most active are Czech group participating in EU DataGrid (EDG) project and IPv6 working group of the Czech research network.

For the more remote future we think about integrating our reflector into Open Grid Services Architecture framework [15] to enable integration with new generation of collaborative Grid environments (e. g. AccessGrid version 2.x [3]). Such integration requires incorporation of Grid service interfaces and Grid security mechanisms into our reflector.

As already mentioned in the section on tunneling between reflectors, another interesting direction of reflector development is implementation of self-organizing and automatic discovery capabilities stemming from ideas of peer-to-peer networking either in pure or hybrid or super-peer mode [8], [16]. This will enable reflectors to create overlay networks that can sustain even partial network disintegration without completely breaking overlay network as it allows the networks of reflectors in the remaining network clouds to work autonomously.

Furthermore we are considering extension of traffic shaping features and congestion control and development of more advanced data transformation processors.

## VI. Acknowledgment

The authors would like to give credits to Assoc. Prof. Luděk Matyska for supporting our work, proof reading this paper, and for stimulating discussions.

## References

[1] R. Wittmann and M. Zitterbart, *Multicast communication: protocols, programming, and applications*, Morgan Kaufmann Publishers, 1999.

[2] P. Galvez, G. Denis, and H. Newman, “Networking, Videoconferencing and Collaborative Environments,” in *Proceedings of CHEP’98*, Chicago, September 1998.

[3] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi, “Access Grid: Immersive Group-to-Group Collaborative Visualization,” in *Proceedings of Immersive Projection Technology*, Ames, Iowa, 2000.

[4] E. Hladká and Z. Salvet, “An Active Network Architecture: Distributed Computer or Transport Medium,” in *Proceedings of ICN 2001*, Berlin, 2001, vol. LNCS 2094, Springer-Verlag.

[5] J. Denemark, P. Holub, and E. Hladká, “RAP - Reflector Administration Protocol,” Tech. Rep. 9/2003, CESNET, 2003.

[6] D. Ed. Crocker and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” RFC 2234, November 1997.

[7] G. A. Roediger, “The Multi-Session Bridge,” `http://www.hep.net/chep98/PDF/230.pdf`.

[8] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, 2001, pp. 329–350.

[9] V. Holer, “Videostram Merging,” Bc. Thesis FI MU, 2003.

[10] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” RFC 1889, January 1996, Obsoleted by RFC 3550.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications,” RFC 3550, July 2003.

[12] T. Rebok and P. Holub, “Synchronizing RTP Packet Reflector,” Tech. Rep. 7/2003, CESNET, 2003.

[13] Tomáš Bouček, “Kryptografické zabezpečení videokonferencí,” M.S. thesis, Military academy Brno, 2002.

[14] Z. Salvet, “Enhanced UDP packet reflector for unfriendly environments,” Tech. Rep. 16/2001, CESNET, 2001.

[15] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[16] B. Yang and H. Garcia-Molina, “Designing a super-peer network,” IEEE International Conference on Data Engineering, 2003.