INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. ● Suite 600 ● Berkeley, California 94704-1198 ● (510) 643-9153 ● FAX (510) 643-7684

JAM: A Java Toolkit for Traffic Analyzing and Monitoring

Andreas März, Roya Ulrich
{maerz, ulrich}@icsi.berkeley.edu
International Computer Science Institute (ICSI)
Suite 600 · 1947 Center St. · Berkeley · California · 94704

TR-96-029

August 1996

Providing reliable multimedia services requires considerable effort with currently available hardware and software. Major difficulties are to cope with the changing quality of service parameters. Network as well as the operating system can handle these requirements and share resources optimal among several active multimedia applications only if proper information about traffic characteristics are available. The traffic characteristics also helps to improve application performance in terms of the execution time and required resources. Therefore, monitoring traffic is an essential step to support performance management in any network. However, because of the dynamic traffic behavior, the on-line monitoring and the on-line analysis of values becomes more important in real-time communication.

In this technical report, a toolkit, called JAM (*Java toolkit for traffic analyzing and monitoring*) is introduced. JAM allows the user to configure a multimedia conference and to collect performance statistics for different protocol layers and provides a graphical user interface for on-line visualization of statistics gained from a running multimedia session.

1 Introduction

Providing reliable multimedia services requires considerable effort with currently available hardware and software. Major difficulties are to cope with the changing quality of service parameters. Traffic requirements and behavior are permanently subject to change as users activities and resource utilization vary in an unpredictable manner. Network as well as the operating system can handle time requirements and share resources optimal among several active multimedia applications only if proper information about traffic characteristics are available [1]. To provide this information the service user has to know what the demand on resources for a certain real-time application is. The information about the traffic generated by an application also helps to improve its performance in terms of the execution time and required resources. Therefore, monitoring traffic is an essential step to support performance management in any network. However, because of the dynamic traffic behavior, the on-line monitoring and the on-line analysis of values becomes more important in real-time communication.

The on-line evaluation of quantitative performance measurements like:

- statistics relating to traffic, for instance: sending and receiving rate, experienced delay and loss in each protocol layer, etc.,
- statistics relating to resources, for instance: CPU and memory utilization, bandwidth usage or wastage caused by each user process, etc.,

help the service user to develop control mechanisms to react to the resource requirements and availabilities. Regarding the measured information, more system resources in terms of processor times, memory, buffer or link capacity, can be allocated to achieve better quality. Reserved, but unused, resources could also be released for other applications.

In this technical report, a toolkit, called JAM (*Java toolkit for traffic analyzing and monitoring*) is introduced. JAM allows the user to configure a multimedia conference and to collect performance statistics for different protocol layers. JAM is implemented by MÄRZ in his Master's Thesis at ICSI [3]. JAM's graphical user interface is implemented in the programming language Java, an object-oriented programming language developed by SUN Microsystems Inc.. This interface provides mechanisms for an on-line visualization of various performance parameters.

The report is organized as follows:

Section 2 provides the background information required to better understand JAM's design concept. Section 3 describes the functionalities provided by JAM and the tools implemented to realize these functionalities are introduced. In Section 4, we give guidelines to install and run JAM successfully. Section 5 summarizes different features of JAM and points out some possible extensions.

2 Background

The basic idea behind the development of JAM was to provide facilities to on-line monitoring "IP over ATM" traffic. Hence, we were concerned with gathering information from different protocol levels from the application to the transmission level.

As an application platform, we used MBone toolkit [2], *vic* and *vat*. The MBone is a transmission standard for multimedia applications (audio, video, text boards) across IP networks. The MBone uses multicasting for data transmission. Anyone connected to the MBone can create a conference, which can be joined by any other user at any time. The MBone tools include the implementation of the lightweight protocol *real-time transport protocol* (RTP) which provides information such as time stamps, sequence number, etc., required for real-time data transmission.

In this section, we will give the necessary background information about RTP and the programming environment Java, helping to better understand the JAM's design concept. All tools implemented to gather statistics about RTP-based traffic from different protocol levels are introduced in Section 3.

2.1 The Real-time Transport Protocol

RTP [4] is proposed by the *Internet engineering task force* (IETF) as a protocol for encoding data with real-time character. RTP is not dependent on particular address formats and can run on top of any transport protocol that the framing mechanisms provide. Furthermore, RTP does not provide any quality of service guarantees. It offers no mechanisms to prevent packet losses nor does it guarantee in-order delivery. RTP offers time stamps and sequence numbers for data packets helping the detection of packet delays and losses. For *vic* and *vat*, RTP runs over UDP/IP.

Different media (e. g. audio and video) are transmitted as separate RTP sessions. There is no direct coupling between both media at the RTP level, however, RTP provides synchronization mechanisms. Major reason for the separation was to allow conference participants to receive only one medium if they choose. The RTP protocol is divided into two subprotocols:

- the data delivery protocol for payload information, which mainly deals with determination of media encoding, framing, error detection, encryption and source identification.
- the *real-time transport control protocol* (RTCP) for control information, which deals with sender identification, receiver feedback and inter-media synchronization.

RTCP packets are transmitted periodically to and from all session participants. Each participant sends RTCP control packets. The time interval between the sending of two consecutive RTCP packets is randomized and adjusted according to the number of participants in the session in order to limit control overhead.

One of the principle aspects of RTCP is to support a distributed monitoring of QoS parameters. Participants in a multimedia session provide quality feedback to all other session members by issuing *sender reports* (SR) or *receiver reports* (RR). If a session participant is an active sender, then an SR is issued periodically and sent to all other participants. If a session participant is a receiver only and has no data to send, then an RR is issued. Both reports have the same data structure, except the SR has additional information about the sending behavior of the session member issuing this SR. This feature of RTCP was used by JAM to gather information about the traffic at the application level.

2.2 The Java Programming Environment

Java [5] is the object-oriented programming language developed by SUN Microsystems Inc. designed to deliver executable content over networks. Java improves the *world wide web* interactivity by providing the ability to display information in the form of animation and interactive applications called *Java applets*. Another form of interactive content provided by Java is that software can be downloaded and executed on any platform with the Java interpretation environment installed.

Highlights of Java can summarized as follows:

- Portability

  Java supports operation in different network environments and on a variety of different hard and software architectures. Hence, the Java compiler does not generate machine code, but a byte code, which is an intermediate code and architecture neutral. This code can be migrated over different platforms as a high level machine code and can be executed by the Java interpreter and the run-time system. The Java interpreter simply executes the byte code on any machine to which the interpreter and the run-time system have been ported.

- Security

  Java does not support pointer operations as used in other high level programming languages like C, C++. This feature leads to robust and secure Java programs. Furthermore, the user does not need to take care of memory management which means there is no need to allocate and free memory for objects anymore. Java uses background garbage collection for memory management. Once memory is allocated for an object, the Java run-time system keeps track of the status of this object and automatically gives up memory if this object is no longer in use. By removing pointer operations and memory allocation at compile time, programmers in Java are unable to move pointers into restricted segments of the system.

  An additional security feature is the byte code generation. The Java run-time system uses a byte code verification process to ensure that the code loaded over the network does not violate any system security restrictions.

- Multi-threading

  Java supports multi-threading which allows that multiple applications run independently from each other at the same time. Multi-threading improves in particular the performance of interactive, multimedia applications. Threads, specified in Java as class objects, comprise an essential part of the Java programming language. The thread class provides a various number of methods to start or stop a thread or get information about the status of the thread.

JAM uses thread objects as control mechanisms for collecting network measurements and for the visualization. All performance parameters can be handled separately by thread objects, which lead to a various number of threads running parallel during an MBone multimedia conference. For the layout of the graphical user interface, JAM uses several objects provided by the Java window toolkit library.

3 The Java Toolkit JAM

At the current developmental stage, JAM allows the user to configure an RTP-based bidirectional MBone session and to collect statistics about incoming and outgoing packets with respect to different protocol layers involved. JAM evaluates and visualizes the gathered information on-line. In this manner, it gives the user a means for observation and performance-driven control of the session.

3.1 RTP Level

For the RTP level, JAM provides the following statistics:

- average payload rate of RTP packets for audio and video transmission, and
- loss rate of RTP packets for audio and video transmission.

Packet information concerning the RTP level is provided by running the C program `getrtcp` that captures RTCP packets from a local port. `getrtcp` was rewritten from the C program `rtpdump` which is used to decode and display packet information for both RTP and RTCP packets. `rtpdump` creates two network sockets and binds these sockets to the port numbers used for the transmission of data and control information. `rtpdump` can be invoked only for monitoring incoming RTP and RTCP packets and captured data do not reach the application process anymore. For example, if a *vic* or *vat* application is started on the local host while simultaneously running `rtpdump` then these applications will not receive any data or control packets from the connected host. However, `getrtcp` allows access to control information about the data traffic during a running MBone multimedia session. `getrtcp` only capture RTCP packets from the socket but does not read any RTP data packet information from the socket. This allows the user to establish an MBone multimedia conference and to gain control information that is exchanged between both hosts during an MBone session.

To calculate statistics about the average payload rate, `getrtcp` reads the time stamp and the sender's octet count data field given in the sender information block. The time stamp indicates the system time when this SR was issued. The sender's octet count shows the total number of payload bytes transmitted in RTP packets by the sender since the beginning of the transmission until the time when this SR was generated. For two consecutive SRs, the average payload data rate may be calculated by dividing the number of payload octets through the time difference between two SRs. The loss rate is given in the lost field of an SR and shows the percentage of RTP data packets lost since the previous SR was sent. The sender of this SR may calculate the number of RTP packets lost by counting the number of incoming RTP packets as well as keeping track of their sequence numbers.

After starting an MBone session, JAM calls `getrtcp` twice for monitoring *vic* and *vat* session separately. `getrtcp` is then initialized with the IP address of the local host and the port number for either video or audio transmission. Once `getrtcp` is invoked it will print the output into a file. The generated files include a short header for the identification of the data contained in the file and can be visualized later. The names of the files depend on the conference name specified by the user. For a conference named `MyTest`, file names will be

- `MyTest.rtp.audio.bw` (bandwidth usage for audio traffic)
- `MyTest.rtp.audio.lr` (loss rate for audio traffic)
- `MyTest.rtp.video.bw` (bandwidth usage for video traffic)
- `MyTest.rtp.video.lr` (loss rate for video traffic)

The visualization information is read from these files on-line.

3.2 UDP/IP Level

JAM gains IP and UDP information by calling the UNIX routine `snoop` that provides a set of options to get packet header information. `snoop` listens on an interface for packets from the network and displays the contents of packet headers in the same order they arrive. These options also allow the user to create a packet filter and to capture only packets that match a certain expression specified by the user. Furthermore, `snoop` presents time stamp accuracy to within 4 microseconds which enables reliable statements concerning bandwidth usage and inter-arrival time. `snoop` also offers a single-line summary output form for captured packets. This saves CPU capacity when evaluating packet information from a temporary file. When starting an MBone multimedia session, the `snoop` routine is invoked if measurements should be taken for either the IP or UDP layer.

For the IP layer, JAM provides the following statistics:

- bandwidth usage,
- distribution of the inter-arrival time, and
- loss rate of IP datagrams.

In this case, `snoop` is being started with the following options:

```
snoop -P -d <device> -td -s <size> "ip and host <source address>"
```

Due to system security `snoop` should be executed only in a special mode. This mode can be specified with the `-P` option which only permits capturing packets addressed to the local host machine over an authorized interface[1]. The interface must be specified by the user and must be given in `<device>` in combination with the `-d` option. The time stamp representation is specified with the option `-td` which will print the time difference since the previous IP packet was received. The `-s` option will truncate each packet after size bytes. This option will capture only the header information from each packet received. For IP packet information, the value size is set to 34. For the IP layer, `snoop` is started with an additional filter expression given in brackets. This filter expression will lead `snoop` to print only IP packets that were sent from the host given in source address. The source address may be given as a symbolic address.

Example:

```
snoop -P -dpa0 -s34 -td "ip and host icsibag0.icsi.berkeley.edu"
```

will lead to the following output if started on host icsibag1.icsi.berkeley.edu:

```
0.14023 icsibag0.icsi.berkeley.edu -> icsibag1.icsi.berkeley.edu IP D=192.6.28.57
S=192.6.28.56 LEN=988, ID=57208
0.05107 icsibag0.icsi.berkeley.edu -> icsibag1.icsi.berkeley.edu IP D=192.6.28.57
S=192.6.28.56 LEN=156, ID=57209
0.00084 icsibag0.icsi.berkeley.edu -> icsibag1.icsi.berkeley.edu IP D=192.6.28.57
S=192.6.28.56 LEN=1039, ID=57210
```

The first column in a output line shows the time stamp followed by the symbolic host names of source and destination. The label `IP` identifies IP datagrams. The next columns show the destination (`D=`) and source (`S=`) addresses. The last two values in a line show the total length of this IP datagram and the identification number. The identification number is a unique identifier for each IP datagram. By using the IP datagram identifier the number of discarded datagrams during transmission may be evaluated. The loss rate is defined as the number of discarded IP datagrams divided by the number of expected IP datagrams. With the time stamp provided by the local system clock and the total length

[1] This has been the ATM interface at the ICSI.

value given in the IP header, the bandwidth usage may be calculated. To get information about the distribution of the inter-arrival time, all IP datagrams with a unique time stamp are summed up.

All measurements are stored into separate files. Again for a conference named `MyTest`, file names will be:

- `MyTest.ip.bw` (bandwidth usage)
- `MyTest.ip.iat` (inter-arrival time)
- `MyTest.ip.lr` (loss rate)

For the UDP layer, JAM provides the following statistics:

- bandwidth usage for audio and video transmission, and
- distribution of inter-arrival time for audio and video transmission.

In order to get UDP protocol information, `snoop` is being started with the options

```
snoop -P -d device -td -s size "udp and host source address and port port"
```

For UDP protocol information, the value size (`-s`) will be set to 42 to capture only UDP headers. Since the port number is given in the UDP header and the MBone tools, *vic* and *vat*, use different ports for data transmission, `snoop` may be used to capture video and audio data separately. This means that `snoop` must be started twice in order to get UDP information about video and audio transmission. `snoop` will print the time stamp, source and destination address and the total length of each UDP datagram.

The bandwidth usage is also calculated separately for video and audio traffic because of the port concept of UDP. The inter-arrival time for UDP datagrams is evaluated analogous to the IP level for both types of traffic. For each call of `snoop`, the output of the program is piped into a special file. Hence, there will be one file to store IP packet information and two separate files to store UDP audio and video information. The file names will include the conference name and either the label `ip` or `udp` referring to the specified protocol `snoop` is invoked with. If `snoop` is invoked to get packet information for the UDP layer, either the label video or audio is attached to the file name in order to distinguish audio and video data files. The files for the example above will be:

- `MyTest.udp.audio.bw` (bandwidth usage for audio traffic)
- `MyTest.udp.audio.iat` (inter-arrival time for audio traffic)
- `MyTest.udp.video.bw` (bandwidth usage for video traffic)
- `MyTest.udp.video.iat` (inter-arrival for video traffic)

4 Installation Guide

Java class library JAM is available via anonymous ftp from:

```
ftp.icsi.berkeley.edu:/pub/tenet/JAM/JAM.tar.gz
```

In following, user guidlines are described to help install and use JAM software successfully.

4.1 System Configuration

JAM version 1.0 is implemented by using the Java Beta 2 version. Java is only running on Solaris workstations. From the MBone video conferencing tool, *vic* version 2.7a32 was used. *vat* was running with the version 4.0a2. Since *vat* 4.0a2 does not use the RTP protocol by default, it must be started using a `-r` flag in order to use the RTP protocol for audio transmission. The variable:

```
Vat.sessionType:  rtp
```

may be set in the .Xresource file in the home directory as an alternative instead of the `-r` flag, to support audio sessions using the RTP protocol. VAT is automatically started with the `-r` flag within JAM when a new MBone conference is started.

Notice that by using JAM over an ATM network the name of the ATM board has to be specified in the variable `INTERFACE` as a String variable in the header of the file `Config.java`. In case this variable has been modified, recomplile the file using the command:

```
javac Config.java
```

at the shell prompt. JAM may then be started by typing the command:

```
java JAM
```

4.2 User Guide

When JAM is invoked, the JAM's **Conference Menu** is shown on the screen (cf. Figure 1) which is the starting point to configure, start, and stop MBone multimedia sessions and to visualize various performance parameters on-line.

- *How to create a new conference*

  Setting up a new conference may be done with the menu choice **Conference-New**. This will pop up the an empty JAM Conference Editor window where a user must specify the conference parameters. A new conference may only be created only if no other conference is active. If another conference is active, then this conference must be closed before a new conference can be created. The section "*How to modify conference*" parameters will describe which parameters must be configured for an MBone multimedia conference.

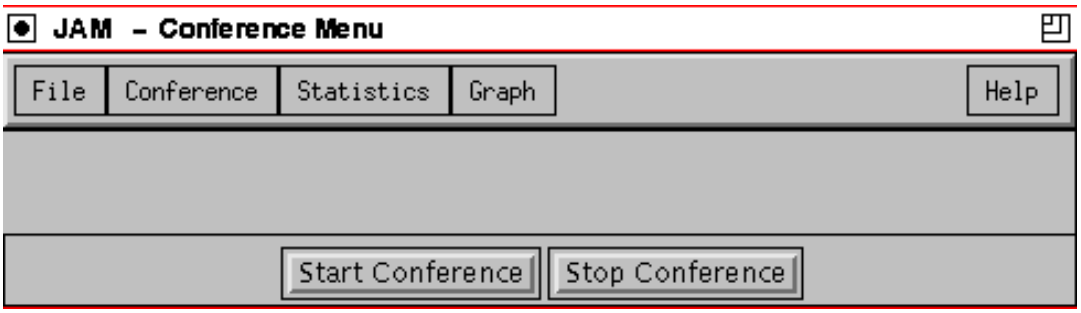

Figure 1: JAM - Conference Menu

- *How to load conference parameters from a file*

  Conference parameters stored in a file may be loaded into the JAM's **Conference Menu**. The user must choose the menu option **File-Open**. This will open a dialog window where the user specifies the file containing the conference parameters. When a file is specified, the conference parameters are automatically read from the file and set as the actual conference parameters. The conference parameters will be automatically displayed in the JAM **Conference Editor**.

- *How to modify conference parameters*

  Conference parameters may be modified within the JAM Conference Editor (cf. Figure 2) by using the menu option **Conference Edit**. The JAM's **Conference Editor** will be automatically shown when conference parameters are loaded from a file or when a new conference is created. Both possibilities have been explained previously.

  In this window the user must specify the name of the conference. The conference name must be given as a string parameter without blanks. This name is used as a unique prefix for all output files generated during a running an MBone multimedia conference. The user specifies the host name of the source and the destination host. Both host names must be given as symbolic host names. Furthermore, the port numbers for video and audio transmission must be given by the user. The default values in JAM for both port numbers are set to 3456 for video transmission and 3458 for audio transmission and may be changed if required. However, the port numbers must be even because the RTP protocol always uses an even port number for data transmission. In the JAM's **Conference Editor** the user may also specify which protocol level information shall be recorded. Either IP, UDP or RTP protocol information may be monitored.

  If all conference parameters are specified, the user may update the complete configuration by clicking the **Set** button. This will modify the conference parameters for the active conference. Updating the conference parameters will not automatically update the file in which the actual conference parameters are stored in case the conference parameters have been loaded from a file. The saving of conference parameters to a file

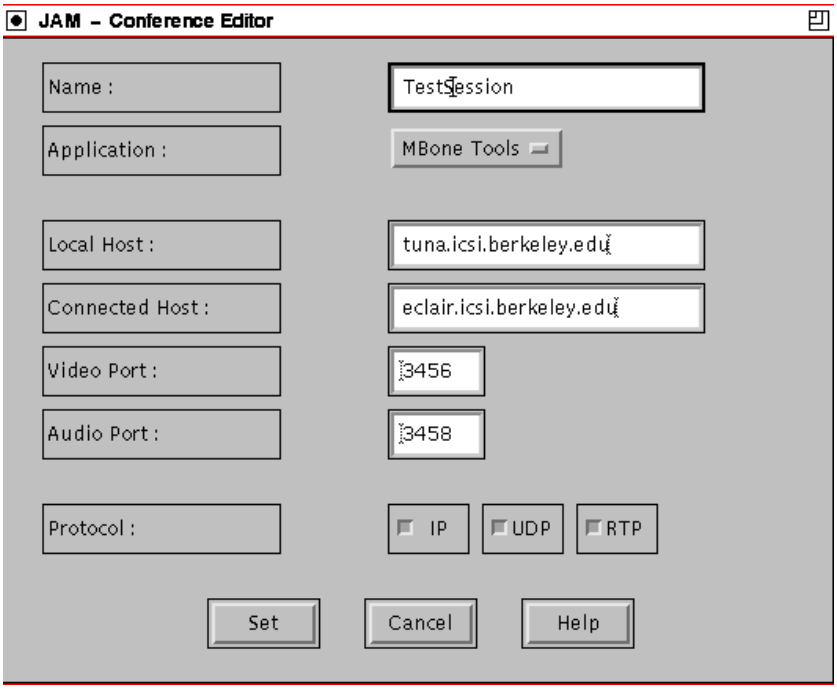

Figure 2: JAM - Conference Editor

will be explained in the section "*How to save conference parameters to a file*". After pressing the **Set** button, the JAM Conference Editor remains visible. The window may be closed by using the **Cancel** button. In case conference parameters have been modified since the last time the **Set** button was pressed, the very last modifications will be discarded.

- *How to save conference parameters to a file*

  A configuration may be saved to a file by choosing the menu option **File-Save**. This will pop up a dialog window where the user may determine a certain file name. Each file only contains parameters for one conference.

- *How to start a multimedia conference*

  After all conference parameters are specified, an MBone conference may be started using the **Start Conference** button in the JAM Conference Menu. In this way, JAM will start the MBone tools *vic* and *vat* to establish the video and audio connection. During the multimedia conference, the user may visualize a various number of performance measurements as specified in the conference parameters. The section "*How to visualize certain performance parameters on-line*" explains how different plots may be illustrated during a running conference.

- *How to visualize certain performance values on-line*

  JAM's **Conference Menu** provides the menu **STATISTICS** where the user may find a various number of submenus which may be used to display certain measurement graphs on-line. The menu **STATISTICS** is divided into submenus corresponding to the supported protocol levels. Plots may only be displayed for protocol levels that are specified in the conference configuration. If a special menu item is selected from the **STATISTICS** menu, the corresponding measurement graph will be shown automatically on the screen.

  For the visualization, JAM supports special features like zooming. By pressing the left mouse button and dragging the mouse in the graph, the plotting range may be altered. This may be used to plot only a part of the data range instead of plotting the complete set of data and to take a more detailed look at a certain graph region. Furthermore, some keyboard commands are supported. By pressing the key `c`, a window will be displayed that contains x- and y-coordinates. These coordinates show the actual x- and y-axes coordinates referring to the mouse position within the graph plot range. They will be changed automatically with each mouse drag. Using this feature, the user may display coordinates of a certain data point. This coordinate window may be closed by pressing the key `c` again. The visible graph plot range may be redrawn by pressing the key `r`. The key `R` will redraw the complete graph by displaying the complete set of data points. The graphs are updated automatically. The user does not need to press a key to update the graph with the most actual measurements taken from the network. Graph windows may be closed and opened any time during a running MBone conference. If a graph window is opened again, it will automatically display the newest measurements.

- *How to stop a multimedia conference*

  A multimedia conference may be stopped by pressing the **Stop Conference** button in the JAM **Conference Menu**. When a running multimedia conference is stopped, JAM will automatically close the audio and video connection.

- *How to close a conference*

  Closing a conference is done with the option **Conference-Close** and will remove all the information contained in the actual conference. This will not affect any measurements made within this conference and will not affect the file in case these conference parameters have been saved to a file earlier. If the conference parameters are not saved to a file, the conference parameters will be lost.

- *How to display a certain performance statistic from a file*

  JAM provides the option to view any graphic from a previous conference which is stored on the file system. Data may be displayed using the **Graph-Display File** option in the JAM's **Conference Menu**. This will open a dialog window where the user may specify the file that should be displayed. After the user has selected a

special file, the data contained in that file is automatically displayed. The previously explained section "*How to visualize certain performance parameters on-line*" shows some features of the general graph layout. These features allow the user to take a more detailed look at a graph.

Figure 3 shows the graphical user interface provided by JAM during an MBone multimedia conference.

5 Conclusion

The on-line traffic monitor, JAM (Java tool for traffic Analyzing and Monitoring allows the user to configure an RTP-based MBone sessions and to collect statistics about incoming and outgoing packets with respect to different protocol layers involved. JAM evaluates and visualizes the gathered information on-line. In this way, it gives the user a means for observation and performance-driven control of the session.

At the current developmental stage, JAM supports the setup of a bidirectional point-to-point MBone session and provides the following performance values separately for each video and audio stream at RTP, UDP and IP level:

- transmission capacity required, and
- end-to-end loss rate experienced.

In addition, the distribution of interarrival times at the IP packets level is evaluated. All measurements can be stored in text files and can be evaluated or visualized by later studies. Figure 3 illustrates an example for a session configuration and measurement using JAM.

We are working on extending JAM by mechanisms to gather information from ATM cell level. Several problems arise at the ATM level because:

- available ATM hard- and software do not provide sufficient means for cell traffic measurements within an accurate time scale, and
- evaluation of data recorded with a fine granularity imposes excessive overheads in processing and storage resources[2].

If an ATM switch provides an interface for standard management protocols like SNMP (simple network management protocol), it is conceivable to implement an agent process collecting proper cell information. The problem still remains that a universal solution cannot be proposed if not all ATM switch products provide at least such a standard interface.

[2]The latest two-days long measurements of ATM cell traffic in BAGNet with a time stamp accuracy of $50ns$ supplied a huge amount of raw data, approximatively 50 Gbyte which can barely be evaluated by available evaluation tools.

An additional focus for future extensions of JAM is integration of data collection related to the operating system status during a certain multimedia session. This will help us to learn more about the system resources being shared between different active real-time processes and about the time being spent in different process states. Understanding these topics is essential for our future research work related to real-time operating systems.

In summary, the information gained by JAM can be used to develop mechanisms to trigger congestion control and to deal with resource management under variable communication conditions. Additionally, the measurements can be used as the input to fit the parameters of a performance model to the key properties of real data stream.

References

[1] Marco Alfano, Rolf Sigle, and Roya Ulrich. A Cooperative Multimedia Environment with User-driven QoS Control. *Proceedings of the 8th IEEE Workshop on Local and Metropolitan Area Networks*, August 1996.

[2] V. Kumar. *MBone: Interactive Multimedia On The Internet.* Macmillan Publishing, 1995. ISBN: 1-56205-397-3.

[3] Andreas März. An ATM Traffic Analyzer in Java. Master's thesis, Universität Erlangen–Nürnberg (IMMD VII) and International Computer Science Institute (ICSI), May 1996.

[4] Schulzrinne, Casner, Frederick, and Jacobson. RTP: A Transport Protocol for Real-Time Applications. Internet draft, Internet Engineering Task Force, 1996.

[5] Sun Microsystems. *All About Java*, 1995. http://java.sun.com/allabout.html.

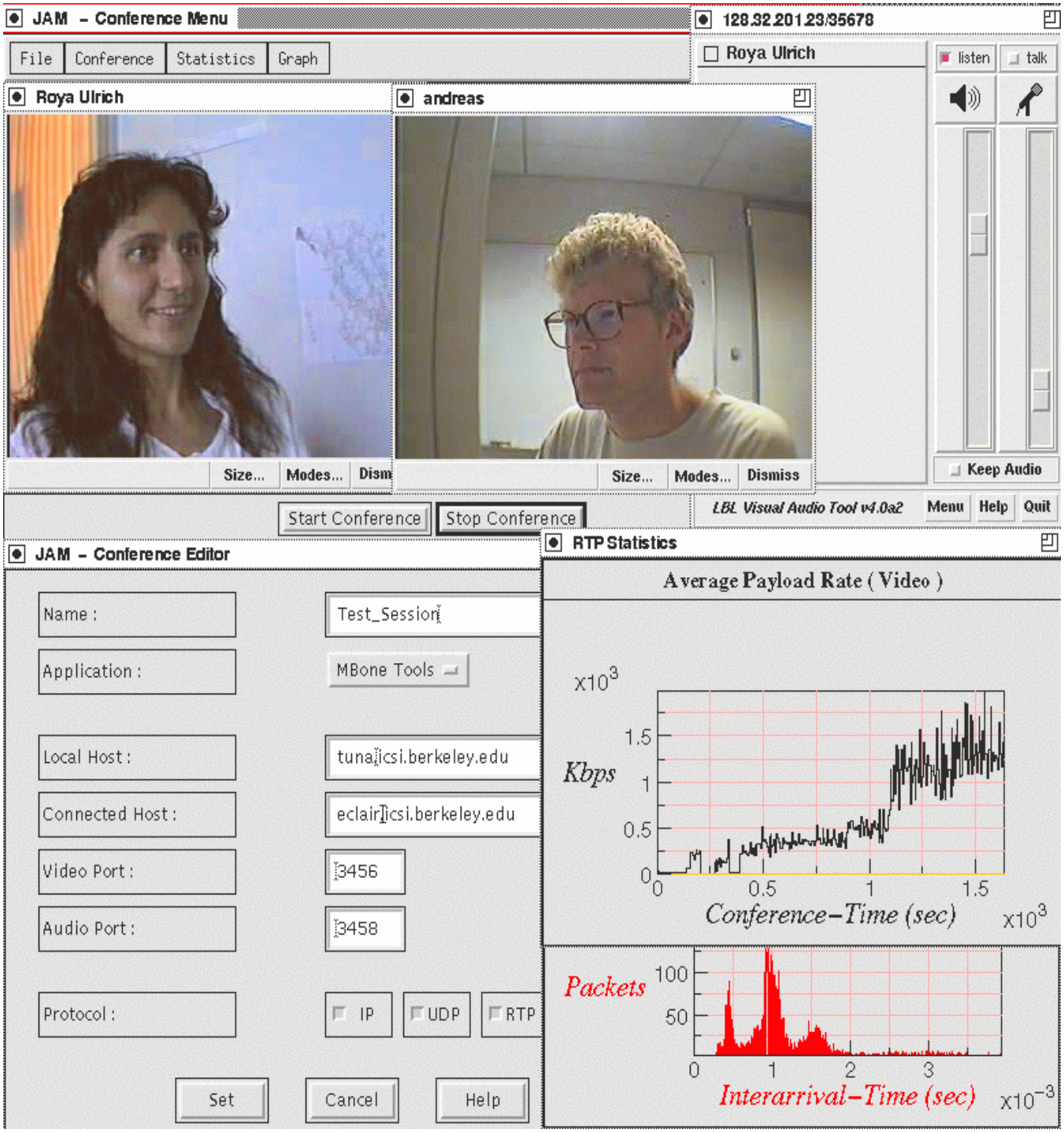

Figure 3: JAM's Layout