AccessGrid-to-Go : Providing AccessGrid access on Personal Digital Assistants

Michael Thorson[1], Jason Leigh[1] (spiff@evl.uic.edu), Gabriel Maajid[2],
Kyoung Park[1], Atul Nayak[1], Paul Salva[2], Shirley Berry[2]

1- Electronic Visualization Laboratory (EVL), University of Illinois at Chicago
2- Chicago Public Schools- Medill Technical and Professional Development Center.

(www.evl.uic.edu/cavern/continuum)

Vic Viewer for PocketPC (VVP) is a software application to decode and display a packetized video stream delivered over IP networks to a PDA running the Microsoft PocketPC operating system. This was first developed in the Spring of 2001 by Michael Thorson at EVL. The software is based on VIC, an open source video conferencing application originally developed at Lawrence Berkley National Laboratory. VIC is the primary means for video distribution over the Access Grid.

The fundamental problem we are attempting to solve is: How does one display dozens of Access Grid video streams on a small screen, over a low bandwidth wireless network, with a small amount of processing power? We believe this is the problem, in general, that needs to be solved for wide spread deployment of portable wireless video conferencing.

At its core, VVP provides compatibility with VIC, video conferencing software originally developed by the Network Research Group at the Lawrence Berkeley National Laboratory in collaboration with the University of California, Berkeley. In some places, particularly in the video decoding, VVP incorporates portions of the VIC source code with little or no modification. Because VVP relies heavily on VIC source, it is important to note critical differences between these two applications:

VIC	VVP
Supports display of multiple video streams in separate windows.	Displays a single video stream in a single window.
Supports a variety of network layers.	Supports IP networks only.
Supports scaling of video to different sizes.	Video is displayed at native size.
Supports 2-way video, can act as a sender and receiver.	Receives video only, no transmitting.
Supports display devices with a variety of color depths and addressing modes.	Supports 16-bit color displays.
Supports the following video codecs: • Xerox Parc Network Video (NV) • Motion JPEG • ITU h.261 • ITU h.263 • Sun Microsystems Cell B • BVC • PVH	Supports the following video codecs: • Xerox Parc Network Video (NV) • Motion JPEG • ITU h.261 • Sun Microsystems Cell B • BVC
Runs on Windows, Unix, Linux.	Runs on PocketPC.

Real-Time Transport Protocol

VIC and VVP both use the Real-Time Transport Protocol (RTP) to receive video streams in the form of User Datagram Packets (UDP) over IP networks. VIC can also send streams according to this protocol. RTP is an Internet-standard protocol for the transport of real-time multimedia data including video. The

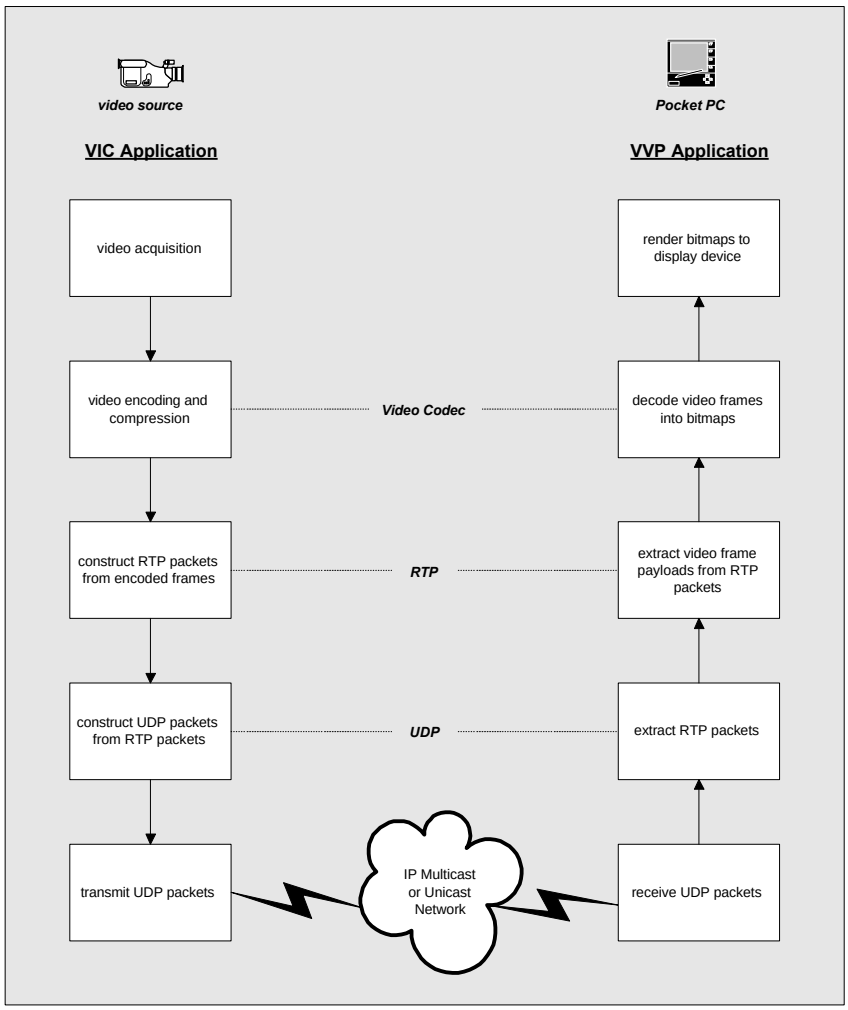
M. Thorson, J. Leigh, G. Maajid, K. Park, A. Nayak, P. Salva, S. Berry, **AccessGrid-to-Go : Providing AccessGrid access on Personal Digital Assistants**, in Proc. Access Grid Retreat, 2002.

RTP protocol does not define video compression formats in particular, but a means to transport time-dependent data in chunks. This lends itself to multimedia applications over UDP. The main functions of the protocol are as follows:

- Carry packetized time-dependent data.
- Provide sequencing and time-stamps so the stream can be accurately played back at the receiver.
- Provide a mechanism for receivers to notify a source of their presence, and their quality of service parameters.
- Defines a means for different sources of multimedia content to be identified.

The protocol is broken down into two parts: data (RTP) and control (RTCP). RTP and RTCP conversations may be carried out over the same IP address, but typically use different port numbers. RTCP carries information about senders and receivers in the session. RTP carries the packets of time-dependent data, video in our case.
RTP and RTCP were developed by the Internet Engineering Task Force (RFC #1889). RTP can be used for a variety of applications. The IETF also specified the RTP Profile for Audio and Video Conferences with Minimal Control (RFC #1890), which declares a list of valid payload types. A payload type for a video stream in RTP is defined according to a compression scheme. VIC was developed to support a handful of these codecs, and VVP supports a subset of these.

The diagram below shows how video packets are transmitted from an RTP source to the VVP application running on a PocketPC.



M. Thorson, J. Leigh, G. Maajid, K. Park, A. Nayak, P. Salva, S. Berry, **AccessGrid-to-Go : Providing AccessGrid access on Personal Digital Assistants**, in Proc. Access Grid Retreat, 2002.

Multicasting and Multi-Source Sessions

Both VIC and VVP support the use of Multicast IP addresses. On networks that support Multicast routing, this allows multiple video sources at different locations to transmit and receive on the same IP address. This is the basis of video-conferencing in VIC. However, VVP is capable of displaying only a single video source at a time, partially due to the form-factor of the PDA device, but also because decoding multiple sources is computationally intensive. VVP is still capable of receiving multiple source traffic over a multicast IP address. At the network interface layer, the VVP application will filter incoming UDP packets on a single source, all other source packets are discarded. There is overhead involved in performing this filtering and the unwanted traffic from other sources will cause performance degradations in the video output quality of VVP. This can be measured in terms of packet loss. Each RTP packet is sequenced so that VVP can monitor how many packets should have been received vs. how many were actually received in a given time period. Qualitatively, packet loss will be detectable as defects in the displayed image or frame hops, where action or movement in the scene does not flow smoothly from one frame to the next.

VVP Proxy

To improve the performance of VVP in multicast and multi-source environments, another software component was developed, the VVP Proxy Application. VVP Proxy is a program that runs on a separate workstation from the PocketPC. It listens to multicast multi-source traffic on behalf of VVP and filters out a single source to forward to VVP. This reduces the amount of overhead VVP spends on filtering UDP traffic. VVP communicates to the Proxy using the RTCP protocol to obtain a list of available sources from the Proxy and to instruct the Proxy which source to filter. The list of available sources (or channels) is presented to the user on the PocketPC for selection.

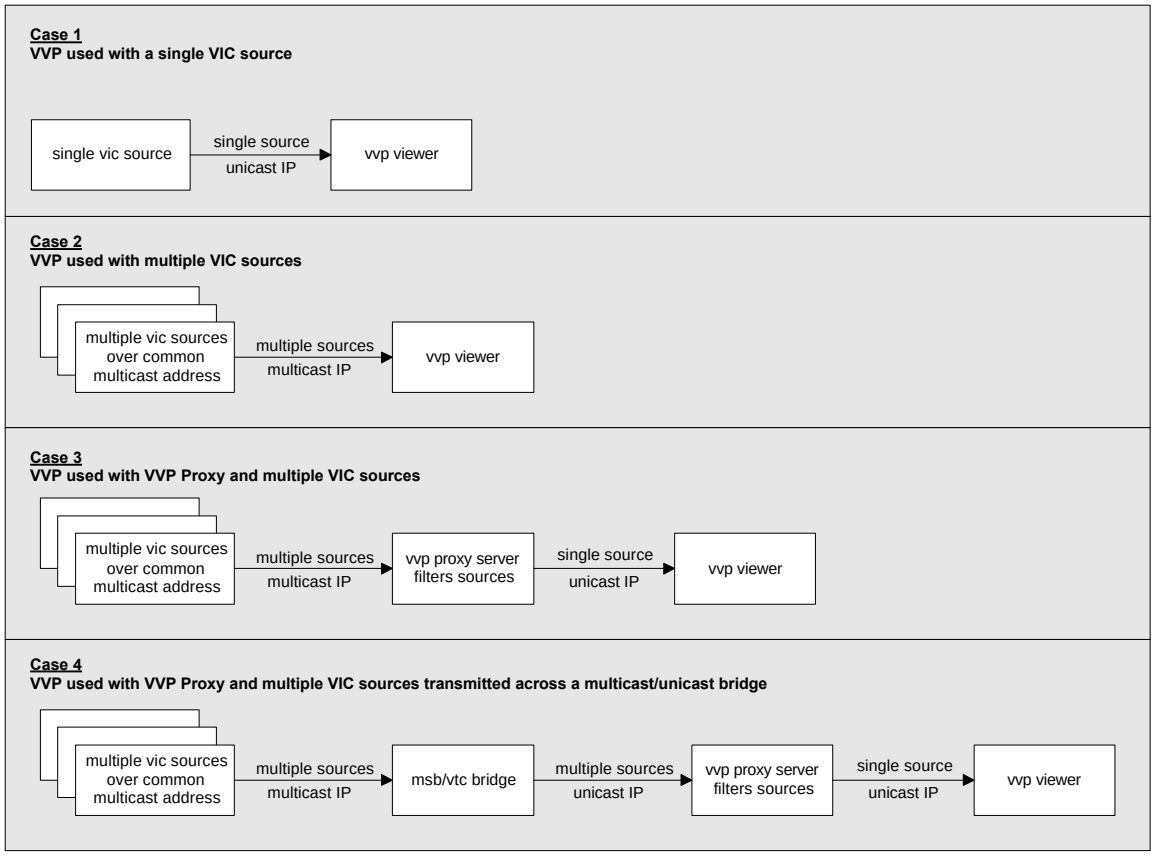
RTCP Application Packets

RTCP provides for different payload types in much the same way as RTP. Normally, VIC is concerned with several types of compound RTCP packets defined by the protocol – Sender Reports, Receiver Reports, and Source Descriptors. VVP is only concerned with Source Descriptors, which it uses to build a list of sources to present to the user. An additional RTCP payload type is defined for Application-dependent data. These Application packets are used by VVP to make filtering requests of the Proxy and also by the Proxy to send source lists to VVP. Otherwise, the filtered video data in the RTP conversation is passed from the Proxy to VVP without modification.

By using application packets with RTCP, VVP can operate with or without the Proxy. The Proxy simply provides a performance enhancement.

Use Cases

With the addition of the Proxy application and the possibility of multiple sources, there are several configurations where VVP might be deployed. The diagram below summarizes these use cases:



Use Case 1 – Single Source VIC

In this case VVP does not need to perform filtering of UDP packets to find a specific source. VVP must be configured with the IP address and port number that the VIC source is using to transmit video. The VIC source will not receive reports from VVP.

Use Case 2 – Multi-Source VIC

VVP will need to perform filtering on the UDP stream to determine which packets belong to the desired source for display. The overhead required to perform this filtering may cause VVP to suffer performance degradations, especially if the combined data rate exceeds 500kbps.

Use Case 3 – Multi-Source VIC through VVP Proxy

VVP and VVP Proxy carry out a minimal conversation over RTCP to exchange information about the available and selected sources. The Proxy handles the filtering so VVP performance improves. Combined stream data rates to the Proxy can be in excess of 4Mbps. A single source should still not exceed 500kbps to perform well with VVP.

Use Case 4 – Multi-Source VIC through Multicast/Unicast Bridge through VVP Proxy

MSB/VTC is part of the Access Grid software. These are separate client and server applications that were designed to bridge areas of the internet that do not support Multicast routing. The stream provided by VTC is identical to the multicast stream provided by the video conferencing multicast address, except that it is forwarded to VVP proxy over a unicast IP address. For VVP and VVP Proxy, this is functionally equivalent to Use Case 3. VTC and VVP Proxy need to be running on the same Windows workstation and the unicast address provided by VTC must be passed to VVP Proxy for listening.

M. Thorson, J. Leigh, G. Maajid, K. Park, A. Nayak, P. Salva, S. Berry, **AccessGrid-to-Go : Providing AccessGrid access on Personal Digital Assistants**, in Proc. Access Grid Retreat, 2002.

Given the variety of ways in which VIC has been extended, there may be many other possible use cases for VVP and VVP Proxy.

VVP Software Design

VVP was written entirely in C++ using the Microsoft eMbedded Visual C++ development environment. Development of the software can be broken down into these main areas:

1. User Interface
2. Network Interface
3. Video Decoding
4. Video Rendering

VVP User Interface Design

The API's provided by the PocketPC operating system are in fact a rich subset of those available in other Microsoft 32-bit Windows operating systems. The message passing mechanisms and device abstraction are also very similar. Programming the user interface for PocketPC is hardly different than programming for desktop Windows versions with a few noteworthy exceptions:

Form factor: the screen dimensions in pixels on a PocketPC device are generally much different than what you will find on a desktop computer, 320x240 pixels is typical. This is the primary difficulty of programming the user interface in the PocketPC, simply how to arrange controls and information in a smaller space without making navigation more complex.

Menus: API's for handling menus are slightly different than in Win32. In the case of VVP there are no custom application adornments to the system-provided menu so this did not present a problem.

Input devices: most PDA devices do not have hardware keyboards. Again, the operating system services shield this to the application level and the programming interface presented to the developer is nearly the same as it would be in the Win32 case.

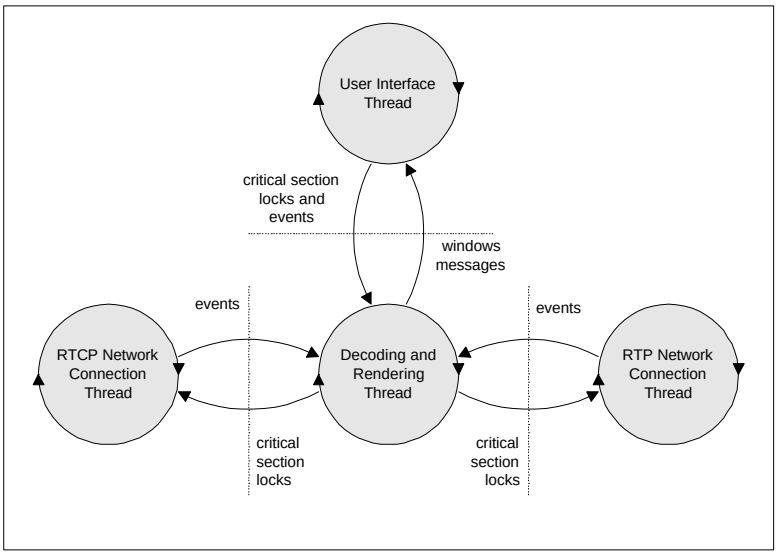
Full Screen: VVP can display video in full screen mode by rotating the frame and using the display in landscape mode. Moving into and out of full screen mode presented a particular challenge. However, PocketPC does provide an API to accomplish this.

VVP Network Interface Design

PocketPC provides the Winsock API, an implementation of Berkeley Sockets for communication over IP networks. There are some important differences between Winsock on the PocketPC and Winsock in other Win32 operating systems that presented problems during development:

1. PocketPC Winsock does not allow application level software to modify the size of the system buffer used for UDP packets. This is a significant shortcoming. The buffer size is only 4096 bytes, which is enough to handle 3 or 4 RTP packets. Depending on the codec being used, a single frame may require more than 4096 bytes. For instance motion JPEG with sufficiently high image detail (quality setting in VIC over about 30) will require more than 4 RTP packets to transmit a single frame.
2. PocketPC Winsock does not provide asynchronous access to sockets driven by system event notification (a useful Microsoft extension to the socket API in Win32). You can however simulate event notification using synchronous or asynchronous sockets and multithreading. VVP uses multithreading and synchronous calls to the Winsock API to provide event driven communication to other layers of the program.

There are separate network connections for RTP data and RTCP control as they use separate port numbers. VVP incorporates an event-driven multithreaded design. Each network connection is given its own thread independent of the user interface. Network connection threads are dedicated to reading UDP packets from Windows sockets and placing them into a secondary buffer. The various threads used in the application along with the mechanisms employed for inter-thread communications and synchronization are shown below.



Due to the limitations of the system-provided UDP buffer (1), it was found during development that UDP packets would be dropped by the operating system if data was being transmitted to VVP at too high a rate. Generally, any time a frame exceeds 4096 bytes, split across multiple UDP packets, part of the frame will be lost. Even with dedicated threads for socket reading, the problem persists. It was found that the UDP stack implementation in PocketPC could not handle high burst rates, average data rates in excess of 500kbps could be sustained, but short bursts exceeding the internal buffer size would result in loss. This was addressed through the use of the Proxy. The Proxy employs a simple method of inserting a millisecond of dead space between each UDP packet that is transmitted to VVP. This has the effect of smoothing the burst rate that occurs when a frame is transmitted. Since the delay between frames exceeds several milliseconds, the average data rate is unaffected by the Proxy and the delay in the stream playback is unnoticeable.

VVP Video Decoding Design

The video decoding and rendering together run on a thread separate from the network interfaces and the user interface. Thus playback is a concurrent, asynchronous process from the transmission being received. Packets are removed from secondary buffering as the decoder is ready for them.

Much of the source code that transforms the RTP payloads into RGB format was taken directly from VIC. The process of extracting only the necessary code from VIC was hindered by the fact that VIC uses a library (TCL/TK) to abstract the platform on which it is running. Fortunately, though, VIC has an object-oriented design which naturally separates the source code into functional units of related C++ classes. A family of classes related to decoding was extracted and the TCL/TK glue was removed. As VIC supports Win32 and the API's are so similar to PocketPC the task of porting these classes to VVP was relatively straightforward.

VVP Video Rendering Design

The rendering portion of VVP, the code that moves buffered RGB frames onto the display device, runs on the same thread as the decoding. VVP uses a video library that was introduced with PocketPC which allows direct access to the video display memory of the PDA. This library is called GAPI (Game API). It was

actually introduced just after PocketPC so not all PocketPC-based PDA devices have this library installed. It can be freely downloaded from Microsoft's web site. It is also included in the VVP installation program.

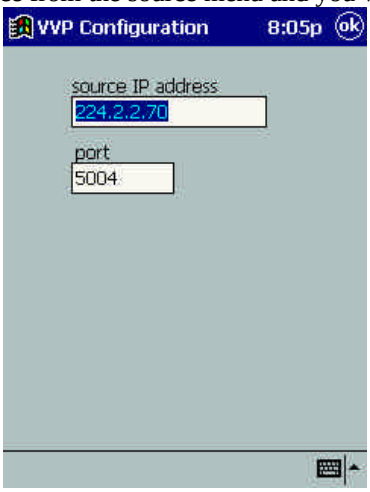
Without GAPI, much of the CPU time used by VVP would be required for rendering using standard graphics device interface (GDI) calls to the operating system. GDI adds an unnecessary layer between the decoded RGB frame and the display device. Frame rendering time can be reduced by over 50% using GAPI. GAPI is required by VVP.

VVP Operation

An installation program is provided for VVP. You will need a PDA device with an Intel Strong ARM or Hitachi SH3 processor and a 16-bit color display. Before running the installation program, make sure your device is cradled and communication has been established with the Microsoft ActiveSync application. VVP was compiled for ARM and SH3 processors and tested specifically on the following PocketPC models:

- Hewlett Packard Jornada 548
- Compaq iPaq H3635

When you launch VVP on the PocketPC, the first thing you will need to do is select the IP address and port of your RTP video source. This may be a unicast or multicast address. It may refer to a direct VIC source or to VVP Proxy. Select configure source from the source menu and you will be presented with this dialog.

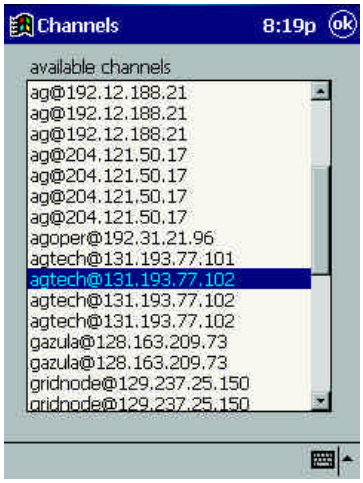


After selecting the IP address, VVP will begin listening for video sources and you will be presented with a list of sources to select the target source (or channel) that VVP will decode and display. The screen below shows an example of the channel list.

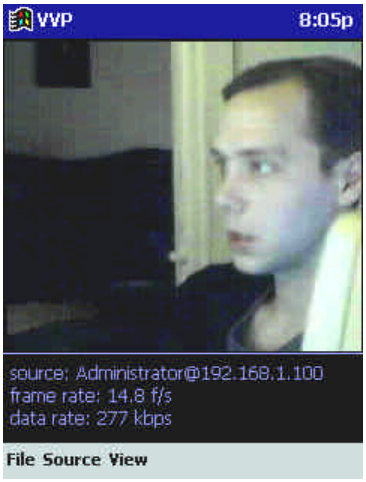
In the sample below, some source names are listed twice. This is an indication that there are multiple streams available from that particular host address. The descriptors listed here are provided by the CNAME parameter defined in the RTCP specification. The CNAME parameter is built from the user logon and host name where the source is originating, and is meant to be unique. It is the only RTCP source description parameter that is required by a source, other parameters describing the source are outlined by the RTCP specification and may or may not be available. VIC uses an algorithm to determine which source description parameters to send in the RTCP conversation. CNAME is sent with the highest frequency. For these reasons, CNAME is presented in the channel list, though there are other parameters that may be available for a given source, they are not parsed from the RTCP conversation or displayed by VVP.

If the source you have specified is a VVP Proxy address, and the Proxy is started, the available channels list should be filled immediately. If you are not using Proxy, it may take a few seconds for VVP to identify the available channel(s). This is due to the lower priority given to the RTCP conversation in the RTP specification. If no channels appear, close the dialog and reopen it by selecting select channel from the Source menu. The list will not refresh until you reopen the dialog. To start decoding and viewing a video

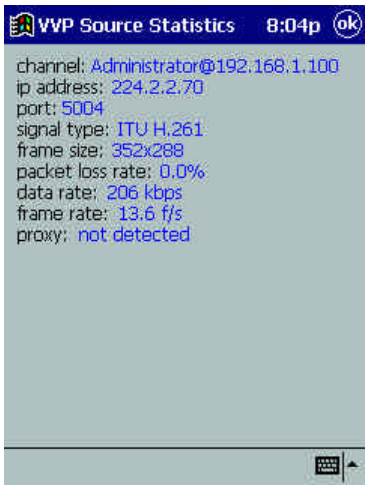
source, select it from the list and tap OK or double-tap the list entry for that source. You can return to this dialog to select a new channel at any time.



Once the video source IP address and channel are selected and locked, the main window of the application will display the video in real-time, along with the channel name and some statistics about the data rate as follows:



The frame will be centered around the available area of the main window. If the frame is larger than this area, it will be clipped and you will see a window into the actual video stream. From the View menu, you can select to display the image full screen. The image will be rotated and fill the entire screen. Again, if the stream is larger than the device display area, you will see a centered view port into the video. In many cases, the native frame size of the VIC source will be at or near 320 x 240 pixels, identical to the display resolution on the PDA's used for testing. In this case you are seeing the full video stream as it is sent. You can determine the actual frame size of the stream by selecting Source Info from the View menu. You will be presented with information and statistics about the stream and RTP session as shown below. You can also use this dialog to determine whether or not the IP source of the RTP session is a direct VIC source or VVP Proxy.



VVP Proxy Software Design

VVP Proxy was written entirely in C++ for Microsoft Windows 32-bit operating systems. It has been tested on Windows NT 4 and Windows 2000 but should also be suitable for Windows 95/98/ME.

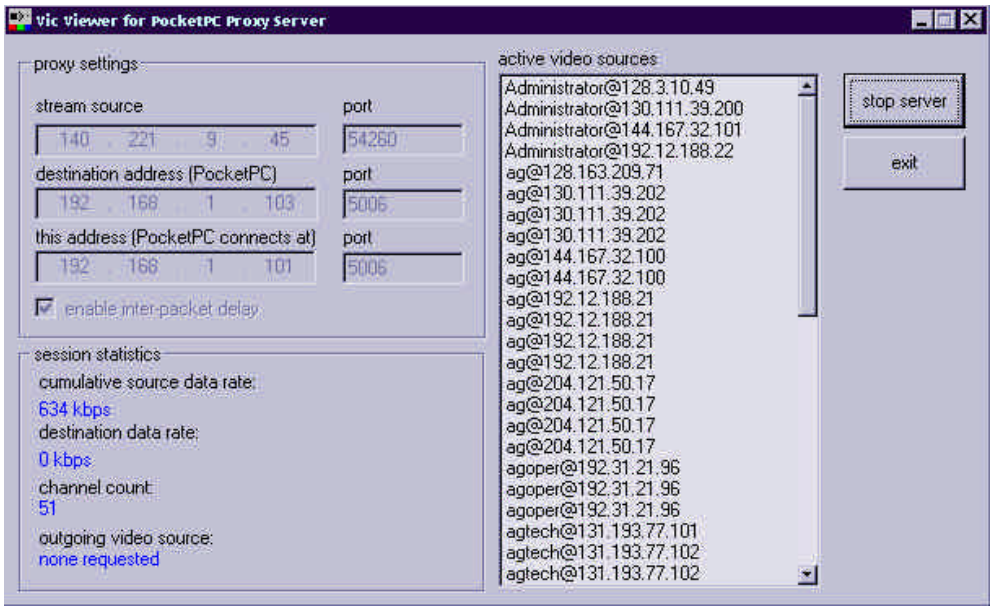
VVP Proxy uses a similar multi-threaded network interface structure to that described for VVP. The primary difference is that VVP Proxy does not decode and render the payloads from the RTP conversation. Instead, it performs filtering at this level and forwards selected packets to VVP.

In VVP Proxy there are 4 network connections instead of 2:

1. The RTP data stream from the video source(s).
2. The RTCP control stream from the video source(s).
3. The filtered RTP stream being sent to VVP.
4. A read/write RTCP conversation between VVP and VVP Proxy for exchanging channel lists and channel selection.

VVP Proxy Operation

Below is a screen capture of the VVP Proxy interface. Upon launching VVP Proxy, you will need to provide the application with the IP address and port number of the RTP session providing the video source(s), the stream source. You will also need to provide the IP address and port for reflection, this is the address the Proxy will forward the selected stream to and should be the IP address of a PocketPC running VVP, the destination address. The local IP address of the Proxy is also displayed for convenience, as this address needs to be entered in the PocketPC VVP source configuration. Once these addresses have been entered, you may start the server.



The list of active video sources will change as new sources are discovered, by design the RTCP conversation occurs at a much lower rate than the video data transmission. Depending on how many sources are involved in the session, it may take several seconds for a source to identify itself in the RTCP conversation, and up to a minute to retrieve all of the source names.

The combined data rate for all sources is displayed. Once a channel has been requested by VVP, the rate of data being transmitted to the VVP destination will also be displayed, along with the name of the source that is being filtered.

Benchmarks and Testing

There are a few key measures of streaming video quality:

- Display rate (fps).
- Delay (for live sources).
- Data rate (kbps).
- Image detail (setting varies among codecs).

Subjectively, I consider the point at which packet loss begins to be the measure of performance. Once packet loss exceeds about 4% with any of the codecs, frame defects are noticeable.

Two of the supported codecs were used for benchmarking: H.261 and Motion JPEG. The other supported codecs are not as widely used and generally to not perform as well. All tests were performed with a highly active scene. This is important because H.261 uses motion compensation, with a highly active scene, more packets are sent. In a more typical scene, the H.261 would be able to achieve higher frame rates with a lower data rate. Tests were conducted using the Proxy with burst rate compensation. Direct VIC sources may achieve slightly lower rates depending on the amount of data in each frame. Frame rates were measured in normal portrait mode, which has a display area of 238 x 206 pixels. The results are summarized below.

Source Configuration	HP Jornada	Compaq iPaq
H.261 video Native frame size 352x288 VVP Proxy VIC source quality setting 10 (default)	Maximum Frame Rate: 9 fps Maximum Data Rate: 260 kbps	Maximum Frame Rate: 15 fps Maximum Data Rate: 500 kbps
Motion jpeg video Native frame size 320x240 VVP Proxy VIC source quality setting 30 (default)	Maximum Frame Rate: 8 fps Maximum Data Rate: 260 kbps	Maximum Frame Rate: 11 fps Maximum Data Rate: 400 kbps

The maximum frame and data rates were determined as the point at which packet loss starts. Loss in Motion JPEG is not as noticeable qualitatively as with H.261, unless it exceeds the ratio of frames to packets, in which case loss can prevent any frames from being constructed. For instance the default quality setting for jpeg requires 3 packets in sequence to construct a frame. Once packet loss approaches 33%, the frame rate will drop to 0. Higher or lower rates can be achieved for either codec by adjusting the image quality setting at the source.

There is a distinct difference between the Jornada and iPaq maximum rates. The difference does not seem to be due to the processing time for Network packets, but rather the time required for rendering frames onto the display device. The Jornada is hitting a ceiling in the number of frames it can draw before it is limited by bandwidth. Frame drawing times were measured for each device and are shown below.

Display Configuration	HP Jornada	Compaq iPaq
Full screen transposed 320 x 240 pixels	142 ms	43 ms
Windowed portrait 238 x 206 pixels	75 ms	30 ms

Conclusion

Though video stream quality and performance are acceptable, the VVP application is at the edge of the PDA device ability to perform. The architectural challenges of working around performance bottlenecks in the PDA hardware and the PocketPC operating system were significant. It is possible that further optimizations on the VVP application can still be made.

In addition, there are a number of ways that VVP could be expanded to provide additional functionality or improved performance. A few obvious ways:

- Add support for more PDA devices, including different display color depths.
- Add support to decode and play live real-time audio synchronized with the video stream.
- Extend the notion of the proxy layer.

There are a number of ways the VVP Proxy application could be enhanced to improve performance on the PocketPC:

M. Thorson, J. Leigh, G. Maajid, K. Park, A. Nayak, P. Salva, S. Berry, **AccessGrid-to-Go : Providing AccessGrid access on Personal Digital Assistants**, in Proc. Access Grid Retreat, 2002.

Interpolation: Interpolate high video rates down to a data rate that is more readily handled by the PocketPC.

Transcoding: Decode non-H.261 formatted video streams and re-encode as H.261. This codec provides the best performance with VVP.

Format Optimization: Ensure that frames within the stream are sized properly for the PocketPC display, decode, resize, and re-encode the stream as necessary.

Descriptive Channels: Though RTCP defines CNAME as the primary identifier of the source, it is often cryptic and not unique. The RTCP protocol provides for other, optional source descriptors. These could be incorporated into the channel list presented to the user.

Multiple Clients: Support multiple VVP clients to the Proxy, filter different channels to each client.

Addendum – Extending VVP to provide multiclient support

To implement multiple client support the proxy uses completion ports. This technology allows for faster response time than when using a single thread per client, by reducing the overhead resulting from context switching. Completion ports use a "pool" of threads that are used when needed. We use overlapped IO. A completion key is associated with a specific IO operation. When it completes, a thread from the "pool" leaves its wait state and responds. The completion key is used to associate the completion of an IO event to code that handles that event.

The proxy joins the multicast group and begins waiting for clients to connect. When clients connect to the proxy their IP addresses are stored in a linked list along with the SSRC of the channel the client requested. When a packet arrives its SSRC is compared to the one stored in the linked list under the clients IP address. If the packet's SSRC matches, the packet is forwarded to theclient. New channels can be selected by the client by requesting that its IP be associated with a new SSRC.

Control communications between the proxy and the client takes place on the RTCP port. This port is equal to the RTP + 1. The proxy listens connection, channel request and termination messages.

Acknowledgments

The virtual reality research, collaborations, and outreach programs at the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago are made possible by major funding from the National Science Foundation (NSF), awards EIA-9802090, EIA-9871058, ANI-9980480, ANI-9730202, and ACI-9418068, as well as NSF Partnerships for Advanced Computational Infrastructure (PACI) cooperative agreement ACI-9619019 to the National Computational Science Alliance. EVL also receives major funding from the US Department of Energy (DOE), awards 99ER25388 and 99ER25405, as well as support from the DOE's Accelerated Strategic Computing Initiative (ASCI) Data and Visualization Corridor program. In addition, EVL receives funding from Pacific Interface on behalf of NTT Optical Network Systems Laboratory in Japan and Microsoft Corporation.

References

The following is a list of URL's to documents describing projects that were referenced in this document, or were otherwise useful in the development of VVP.

Internet Engineering Task Force (IETF). http://www.ietf.org/

IETF Audio/Video Transport Working Group (avt) http://www.ietf.org/html.charters/avt-charter.html

Real-Time Transport Protocol (RTP), RFC #1889. http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-new-08.txt

M. Thorson, J. Leigh, G. Maajid, K. Park, A. Nayak, P. Salva, S. Berry, **AccessGrid-to-Go : Providing AccessGrid access on Personal Digital Assistants**, in Proc. Access Grid Retreat, 2002.

RTP FAQ. http://www.cs.columbia.edu/~hgs/rtp/

RTP Profile for Audio and Video Conferences with Minimal Control, RFC #1890 http://www.ietf.org/rfc/rfc1890.txt

Multimedia Integrated Conferencing in Europe, the MICE Project Home Page.
http://www-mice.cs.ucl.ac.uk/multimedia/projects/mice/mice_home.html

University College London (UCL), Networked Multimedia Research Group, VIC sources and binaries for all available platforms. This is part of the MICE project.
http://www-mice.cs.ucl.ac.uk/multimedia/software/

VIC FAQ at UCL.
http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/faq.htm

Network Research Group at Lawrence Berkley National Laboratory, the origins of vic.
http://www-nrg.ee.lbl.gov/vic/

Microsoft Bay Area Research Center (BARC): MBONE and IP Multicast.
http://www.research.microsoft.com/research/BARC/mbone/

Access Grid, Argonne National Laboratory.
http://www-fp.mcs.anl.gov/fl/accessgrid/default.htm

Microsoft PocketPC, Main Page.
http://www.microsoft.com/mobile/pocketpc/default.asp

Microsoft PocketPC, Developer Page.http://www.microsoft.com/mobile/developer/default.asp

Microsoft eMbedded Visual Tools 3.0.
http://www.microsoft.com/mobile/downloads/emvt30.asp