Optimized mobile thin clients through a MPEG-4 BiFS semantic remote display framework

P. Simoens[1,2] – B. Joveski[3] – L. Gardenghi[3]– I.J. Marshall[4] – B. Vankeirsbilck[2] – M. Mitrea[3] – F. Prêteux[5] – F. De Turck[2] – B. Dhoedt[2]

[1] HoGent – Dept. INWE
Schoonmeersen 52
9000 Gent, BELGIUM

[2] Ghent University – IBBT, Dept. INTEC
Gaston Crommenlaan 8 bus 201
9050 Gent, BELGIUM

[3] Institut Telecom
ARTEMIS Dept.
9 Rue Charles Fourier
91000 Evry, FRANCE

[4] Prologue
ZA de Courtaboeuf
12 Avenue des Tropiques
91943 Les Ulis CEDEX, FRANCE

[5] MINES ParisTech
60 Boulevard Saint-Michel
75272 ParisCedex 06
FRANCE

Abstract. According to the thin client computing principle, the user interface is physically separated from the application logic. In practice only a viewer component is executed on the client device, rendering the display updates received from the distant application server and capturing the user interaction. Existing remote display frameworks are not optimized to encode the complex scenes of modern applications, which are composed of objects with very diverse graphical characteristics. In order to tackle this challenge, we propose to transfer to the client, in addition to the binary encoded objects, semantic information about the characteristics of each object. Through this semantic knowledge, the client is enabled to react autonomously on user input and does not have to wait for the display update from the server. Resulting in a reduction of the interaction latency and a mitigation of the bursty remote display traffic pattern, the presented framework is of particular interest in a wireless context, where the bandwidth is limited and expensive. In this paper, we describe a generic architecture of a semantic remote display framework. Furthermore, we have developed a prototype using the MPEG-4 Binary Format for Scenes to convey the semantic information to the client. We experimentally compare the bandwidth consumption of MPEG-4 BiFS with existing, non-semantic, remote display frameworks. In a text editing scenario, we realize an average reduction of 23 % of the data peaks that are observed in remote display protocol traffic.

1 Introduction

In a thin client computing architecture, application and data processing are offloaded to remote servers, while the functionality of the client device is essentially limited to user interface functionalities. Nowadays, the web hosts increasingly powerful computing resources and has evolved to a ubiquitous computer, offering applications ranging from simple word processors, over all-encompassing enterprise resource planning suites to 3D games [26]. Although recent advances in hardware miniaturization have drastically increased the processing power of mobile (smartphone, pda) and portable devices (laptop), the principle of thin client computing remains of particular interest in a mobile context. A first advantage is that applications need not to be tailored individually for each mobile platform. Second, mobile thin client computing allows companies to better protect their classified data. By running office applications on a centralized company server, mobile employees can ubiquitously access all necessary data while at the same time the confidential and valuable data stays on the trusted and secured central server [21]. For example, no data can get lost owing to the theft of a mobile device of one of the employees. Third, beyond the conventional office applications, mobile thin client computing allows to provide resource demanding applications to mobile users. Despite the advances in mobile hardware, mobile phones still lack processing and storage resources to execute 3D virtual environments that require advanced graphical hardware [2], or applications that operate on large data sets, such as medical imaging applications [17].

In the thin client computing principle the mobile device only runs a viewer application that acts as a remote interface to applications running on distant servers. This viewer transfers user input to the server, and renders the display updates received from the server. To encode the display updates, existing remote display solutions typically use low-level graphic instructions or video codecs. The first approach is adopted by conventional remote display architectures. For instance, Citrix XenApp [3] and MicroSoft Remote Desktop Services [23] apply elementary drawing commands and images, whereas Virtual Network Computing [29] updates the display by filling rectangular screen areas with bitmaps. These low-level graphic instructions are the most efficient way to remotely display applications that only update a small portion of the display and have a slow refresh rate, such as office applications. In turn, video codecs are more convenient when the graphics exhibit a high level of detail and large parts of the display are frequently updated, such as in multimedia or 3D applications. This approach is mainly used for multimedia applications and remote 3D rendering, such as virtual 3D environments or 3D medical imaging applications [25, 27].
All existing remote display solutions encode the complete display using a single output encoding format. This encoding format is however only optimized for a specific subset of applications. Using the inappropriate format to compress application graphics leads to high bandwidth requirements and degraded visual experience, *e.g.* because static displays are encoded as video frames or because text characters are encoded by lossy image compression techniques [34]. To support a wider range of applications with a single remote display framework, hybrid approaches have been presented that integrate multiple encoding formats. For example, Citrix’ Speedscreen Acceleration forwards video streams in their original format to the client, while the other parts of the display are still encoded through the Citrix’ proprietary protocol. This approach is only possible for video streams for

which the appropriate video codec is installed at the thin client. Similarly, Tan et al. [36] divide the display in low- and high-motion regions, that are encoded respectively by means of VNC drawing primitives and MPEG-4 AVC (a.k.a. H.264) frames.
These hybrid techniques provide no adequate solution to compress the complex scenes of contemporary applications, comprising objects with highly diverse graphic characteristics, such as text, images, widgets and embedded audio and video. A better approach is to encode each object individually, in its own optimal encoding format. However, this requires that the client is provided with the appropriate information on the different objects, such as their position and encoding format, in addition to the binary encoded objects itself. Within the context of this paper, we refer to this information as *semantic* information and introduce the concept of semantic remote display. As illustrated in Figure 1, the semantic knowledge enables the client to consider the scene as a collection of objects, instead of an elementary pixel buffer. The nature of these objects can be very diverse, examples include menu windows, dialog windows, images, character strings or video streams. In this paper, we demonstrate that the enhanced view of the client on the displayed scene can be exploited to optimize the thin client user experience in terms of interaction latency and bandwidth. These challenges are of particular importance in a wireless context, due to interference and fading effects. As a consequence, the presented semantic remote framework is targeting all types of wireless portable devices acting as a thin client, including laptops, smartphones and pda's.

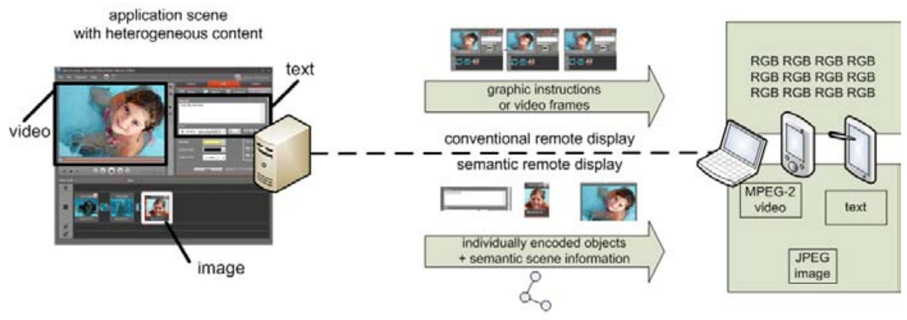

Fig. 1. Traditional remote display solutions typically encode the complete display using the same output encoding format, whereas semantic remote display protocols transfer individual objects in their own encoding format, as well as the objects' characteristics, *e.g.* potential user interaction.

The contribution of this paper is twofold. First, we introduce a generic semantic remote display architecture. Compared with existing, non-semantic remote display solutions, the server side component of the architecture is enhanced with a Scene State Manager, orchestrating the transfer of semantic information to the client. We have developed a prototype of this architecture, using the semantically rich MPEG-4 Binary Format for Scenes (BiFS). In previous work [13,24], we have already demonstrated the viability of MPEG-4 BiFS as compression technology for objects with diverse graphical characteristics, in terms of bandwidth and visual quality. Compared to other technologies for the representation of heterogeneous content, such as Flash and SMIL/SVG, BiFS has the particular advantage that it allows for dynamic updates on the scene description, i.e. updates that are generated at the server during the session. Furthermore, it is based on public standards, and thus avoids proprietary solutions such as Flash, that may lead to vendor lock-in. The BiFS principles have been optimized for thin clients, resulting in the Lightweight Application Scene Representation (LASeR) [18]. However, BiFS seems to be more future proof, owing to its more complex scene description possibilities, as well as the possibility to describe 3D scenes [12].

Second, we provide the client with additional semantic information on how the user can manipulate each object in the scene. This allows the client to react autonomously to user events, *e.g.* by displaying a cached object, instead of having to wait for the server feedback. Client side handling of user input improves the interaction latency and reduces the amount of data that is sent by the server to the client. Compared with existing cache mechanisms such as in HTTP, the novelty of our approach is that we not only cache objects, but also the possible user interactions with the objects in the scene, as well as how these objects should be reorganized after some user interaction. This is a more fine-grained approach, allowing to update smaller parts of the scene (not always complete objects), which in turn results in bandwidth and latency reductions.

The remainder of this paper is structured as follows. In section 2, we elaborate on how semantic information can be exploited to enable client side handling of user input. To manage this semantic information, additional components are required at the server side of a semantic remote display. Therefore, we detail our generic architecture of a semantic remote display framework in section 3. We present our MPEG-4 BiFS based prototype of this architecture in section 4, and detail how client side user input handling is enabled by delivering the appropriate semantic information to the client. In section 5, we experimentally compare our approach with existing, non-semantic, remote display frameworks and quantify the reduction of the amount of data owing to the handling of user input at the client. Related work is explored in section 6.

2 Client side handling of user input

After the user has generated some input, it takes at least one network roundtrip time to transfer this information to the server and the resulting display update back to the client. Users tolerate interaction latencies up to 80 ms for gaming [4] and up to 150 ms for office applications [38]. Wireless links, however, often introduce propagation delays in the order of tens of milliseconds [39, 40] and the total latency is further increased by client and server processing, router queuing, firewall processing etc. Furthermore, the display updates that are triggered by user input, often contain a large amount of data that needs to be transferred to the client in a short interval. This results in a bursty traffic pattern, often requiring an instantaneous bandwidth that is much higher than the average bandwidth availability [35].

Semantic remote display provides a solution to reduce the interaction latency and mitigate the bursty traffic pattern. As illustrated in Figure 1, semantic information allows the client to identify the individual objects in the scene. By providing additional information to the client on how the user can actually manipulate these objects, the client can react autonomously to user input and immediately take the appropriate action. For instance, when the user clicks on the “File” menu item object in a text editor, the client knows that an additional object needs to be displayed, *i.e.* the menu that needs to be opened. When cached at the client, this menu can be displayed immediately and the interaction latency is significantly reduced. Furthermore, the server only needs to transmit a differential update to correct or complete the actions undertaken by the client, which will mitigate the data peaks observed in remote display protocol traffic. Retaking the example of the “File” menu, this would be the list with recently opened files.

When only graphic information is delivered to the client, client side user input handling cannot be adequately realized. Consequently, most non-semantic remote display protocols do not support the handling of user input at the client. An exception is Citrix, which has leveraged its architecture with the Speedscreen Latency Reduction feature. When this option is enabled, the thin client will predict the server response to key strokes and immediately show the character on the screen. The mechanism is however limited, because the character is often shown in a different font [32]. After the feedback of the server is received, the font on the screen is corrected.

To enable client side user input handling, the client needs to have accurate information on the potential user interactions and the appropriate actions to be taken subsequently. For each application, we model this information in a scene state diagram that is composed of different scene states. A scene state is defined as a specific configuration of visible objects in the scene. In each scene state, the user can interact with some of the objects, *e.g.* by clicking on it. In the diagram, these objects are marked as transition triggers of the scene state. When the user interacts with one of these trigger objects, a scene transition is started. Typically, a scene transition involves two types of operations: creation and deletion of visible scene objects, and drawing operations to modify the graphical characteristics of these objects. After the transition, a new subset of objects is visible on the screen and the application has entered a new scene state.

A sample scene state diagram for a text editing application is shown in Figure 2. In this example, the scene state is uniquely determined by its visible widgets, menus and pop-up dialog windows. When the user starts the application, he is presented the default screen. All items in the top menu bar are marked as transition triggers. When the user clicks on one of these items, a menu becomes visible. The corresponding scene transition involves the creation of this menu object, as well as drawing instructions to present the individual menu items.

If the client is able to identify the different scene states, as well as the associated transition trigger objects, it can perform the scene transition autonomously. However, intercepting semantic information from the application, constructing the scene state diagram and transferring this information to the client, requires that the functionality of the server side component of remote display frameworks is leveraged with additional components. Furthermore, compared to conventional non-semantic frameworks, the architecture needs to provide the required flexibility to deliver to the client individually encoded objects instead of drawing commands or video frames, as well as semantic information. In the next section, we present a generic architecture of a semantic remote display framework that meets these design requirements.

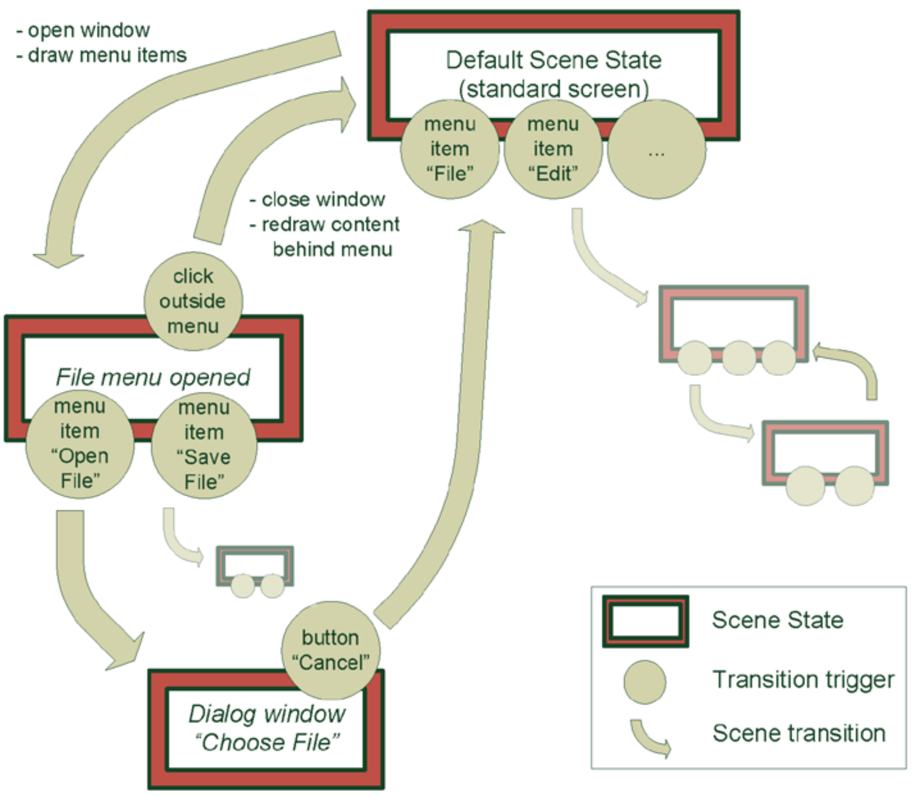

Fig. 2. Scene state diagram of a text editing application.

3 Related work

The framework presented in this paper enables the client to handle user input locally for remote applications. Related work that has been presented in literature was mainly focused on the delivery and caching of dynamic content in web services.

In [19], the authors enhance the declarative SMIL language with an external data model, to adapt the presentation of a web service in a more flexible way to the user input. The data model contains user-defined state and allows to mitigate the lack of support for interactive web based applications that is common to all declarative languages such as HTML, SMIL or SVG. By contrast, the BiFS format that is used in our framework has inherent support for dynamic updates from the server, triggered by user input. Furthermore, our framework is targeting applications that are currently installed on desktops, instead of web services.

In [10], the authors present a combined content adaptation and caching scheme for displaying HTML content on mobile devices. After splitting the web page in HTML content blocks, each block is adapted to the size of the mobile device display. Furthermore, an integrated content block cache helps in reducing the mobile client latency to fetch HTML content from a web server. Similarly, in [5], a model is presented that tries to predict the future navigation commands during an interactive browsing session to optimize the client cache of mobile devices. The presented work focuses on improving the remote browsing of large scale images, which is challenging because of the small size of mobile displays and the cost and latency associated to data transmission. The presented caching algorithms are however designed for JPEG2000 images and cannot directly be applied to the case of user actions that is the scope of the current paper.

Another direction of related work can be found in the exploitation of the distinctive feature of MPEG-4 BiFS to describe a displayed scene in terms of objects and the potential user interactions with these objects. Kim et al. [15] have developed an MPEG-4 object streaming server system that streams individual objects upon user request. The presented system provides a concrete implementation of how objects can be streamed over multiple remote display channels. Consequently, it is complementary to our work, which focuses on how these objects should be intercepted from the application.

Khin et al. [14] have developed an Interaction Manager that analyzes the user input and determines the appropriate modifications to the BiFS scene. The Interaction Manager supports more complicated operations to react on user input, such as converting and encoding user input to the appropriate output format for online quiz applications. The proposed architecture may however be difficult to deploy on thin mobile devices, because additional components need to be installed on the client, consuming additional memory and CPU. Furthermore, a detailed experimental evaluation is missing, making it difficult to assess the footprint of the installed components in terms of memory and CPU.

4 Architectural design of a semantic remote display framework

Any remote display framework contains a client side viewer and a server side component. The viewer application imposes no stringent design requirements, as it functionality is limited to transferring user input to the server and rendering the display updates from the server. Therefore, we will focus the discussion in this section on the server side component of a semantic remote display architecture, which is presented in Figure 3.

The server side component is located between the application logic and the network communication interface and is composed of two logical layers. The upper layer, the Content Manager, intercepts the content generated by the application and converts it to the appropriate output format. Furthermore, it manages the semantic scene information. Both the converted graphical content as the semantic information are delivered to the client through one of the communication channels in the lower layer of the architecture. In the upstream direction, user events are received by the user event handler, after possible conversion in the Content Manager, and delivered to the application.

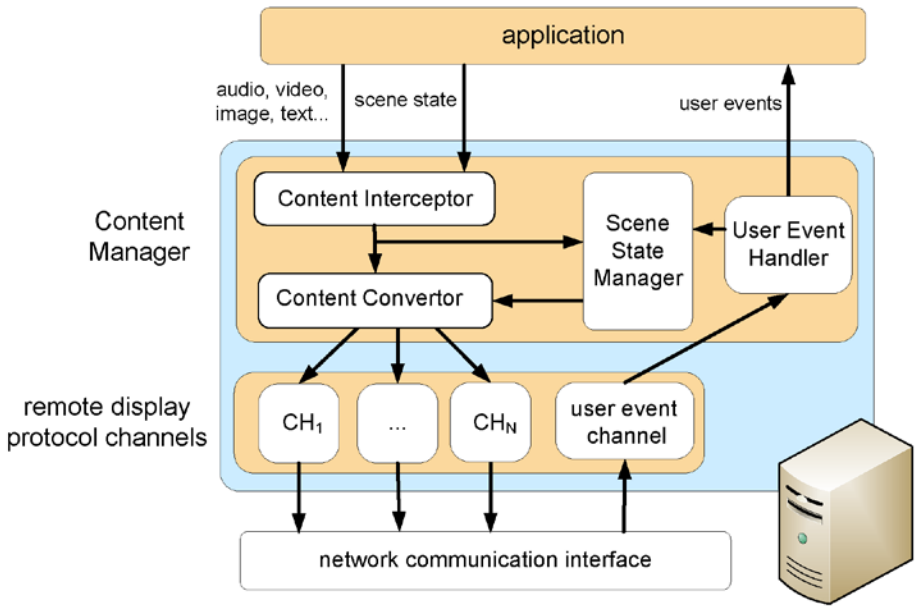

Fig. 3. Server side component of a semantic remote display architecture. The Scene State Manager (SSM) manages the semantic scene information and determines the appropriate output encoding format for each content type.

The architecture is composed of the following components:

- Content Interceptor: captures various types of content generated by the application, such as video streams and drawing commands, as well as semantic scene information. Graphical information is directly forwarded to the Content Convertor, whereas semantic information is delivered to the Scene State Manager (SSM). The application content can be intercepted at different layers of the rendering stack. The actual interception point is a tradeoff between general applicability of the remote display framework and the availability of semantic information. For example, an architecture that intercepts at the pixel level can be used across almost all platforms, but no semantic information will be captured. On the other hand, if the graphics are intercepted at a higher layer, *e.g.* at the level of graphical libraries such as Xlib, very detailed semantic information is intercepted, but the interceptor can only be used with applications that are built on this library.
- Content Convertor: converts the input captured by the content interceptors to the appropriate output format, as instructed by the SSM. It encodes both graphic content and semantic information that is delivered by the SSM. The supported conversions depend on the codecs that are installed on the client.
- Protocol channel: is a logical connection between the server side and client side component of the architecture. Each channel transfers a specific content type and is mapped, possibly together with other channels, on an underlying network connection. This allows to provide each channel with the appropriate Quality of Service from the network. Channel examples include embedded video streams that are transported over UDP, or a dedicated channel for user events that requires reliable transmission over TCP.
- User event handler: is responsible for delivering user events to the appropriate application. When the user is concurrently running multiple applications, it queries the SSM to determine the application to deliver the event to.
- Scene State Manager: is the heart of the Content Manager and has a double functionality. First, this component orchestrates the conversion of intercepted graphic content to the appropriate encoding format. Second, it manages and interprets the semantic information captured from the application. This information is used to determine the appropriate output format for the intercepted content. For example, it can detect video streams in the scene, and forward the original bit stream without conversion to one of the output remote display protocol channels. Relevant semantic information is communicated to the client.

5 Prototype implementation

The multiple logical remote display protocol channels and the orchestrating Scene State Manager ensure that the architecture presented in the previous section offers the required functionality to support the remote display of modern applications. In this section, we elaborate on the implementation details of the prototype we have developed. Furthermore, we detail how the Scene State Manager supports the client side handling of user input.

5.1 Overall implementation details

To demonstrate the concept of semantic remote display frameworks, we have implemented the prototype that is presented in Figure 4.

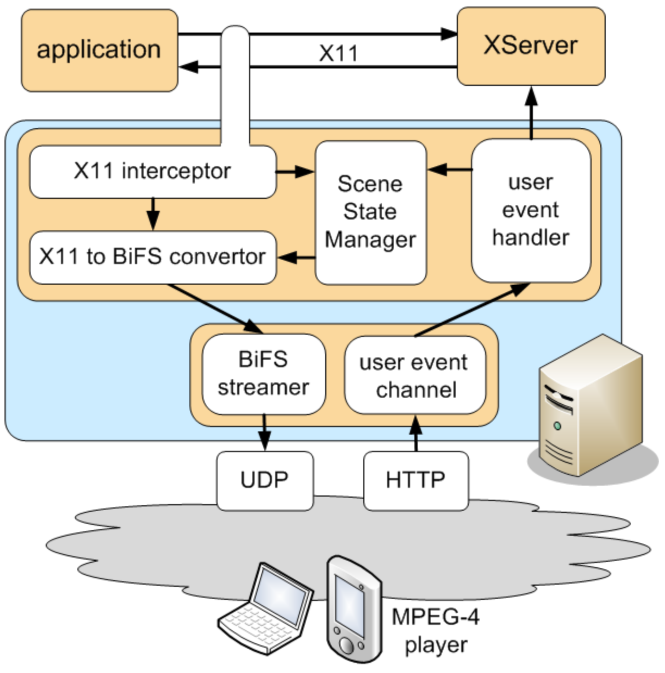

Fig. 4. Overview of the implemented prototype. The Scene State Manager decides which X11 instructions are transcoded to the MPEG-4 BiFS.

The prototype was developed in a Linux environment, where applications typically render their graphics through the X Window System. Applications communicate to the rendering XServer by means of the X11 protocol. This protocol is an appropriate interception point, as it conveys both drawing information, *e.g.* to draw a rectangle or to display an image, and semantic information, *e.g.* to map a window on the screen. Furthermore, the X11 protocol can be transparently intercepted, because the application connects to the XServer through a local Unix socket. The X11 interceptor places itself between the application and the XServer by presenting a similar Unix socket to the applications. Through an elementary modification of the “DISPLAY” environment variable, the applications are instructed to connect to this socket instead of to the original socket opened by the XServer. The X11 interceptor opens a

connection to the XServer and forwards the unmodified X11 data, thus ensuring complete transparency to both application and XServer.
The Scene State Manager tracks all X11 protocol traffic and decides which instructions are forwarded to the X11toBiFS convertor. Semantic protocol messages, *e.g.* when a new window object is created, are used to update its internal scene state information. More information on the internal operation of the SSM is provided in section 4.2.
The X11toBIFSconvertor component receives X11 instructions from the X11 interceptor and additional semantic instructions from the SSM, and converts this to the MPEG-4 Binary Format for Scenes (MPEG-4 BiFS) [11]. BiFS is an MPEG-4 scene description language that combines sufficient semantic expression power with streaming capabilities. It is designed for interactive rich-media services that include text, audio, video, 2D and 3D graphics. The format allows to describe all content by a scene graph, providing a hierarchical and integrated representation of audio, video and graphical components. A distinctive feature of BiFS is that only the scene graph is binary encoded, whereas each object can remain encoded with its own optimal coding scheme. Lastly, BiFS provides the means to stream this encoded scene information to the client and to provide real-time updates to the scene tree. More details on the MPEG-4 Binary Format for Scenes can be found in [33].
Internally, the X11toBIFSconvertor contains a specific conversion function for all X11 protocol message types and the semantic instructions from the SSM. It is built on the GPAC libraries [20] that support the BiFS encoding. Both scene state information, *e.g.* on the size and the position of the created windows, and actual drawing commands, *e.g.* to put an image in the scene, can be converted. At the end of the chain, the BiFS encoded content is streamed to the client over UDP. The BiFS streaming is realized by a modified version of the Live 555 Streaming Media library [22].
In the upstream direction, the client transfers the user events as http requests. These user events are posted on the XServer through the standard Linux *uinput* kernel module. The XServer cannot discriminate between these user events and user events that would be generated by a keyboard or mouse attached to the server machine, thus further ensuring the complete transparency to the application and the XServer.
At the client, the GPAC player has been used. This is a standard and low complexity MPEG-4 player that is supported on both Windows and Linux platforms.

5.2 Internal operation of the SSM

In section 2, we have explained how the possible interactions of the user can be modeled by means of a scene state diagram. In the current section, we provide concrete details on how this diagram is constructed inside the Scene State Manager, and how the SSM uses this information to inform the client on potential user interactions. To support the discussion, Figure 5 visualizes the communication between the server components, as well as the internal scene state diagram of the SSM, for the sample case of a text editor.

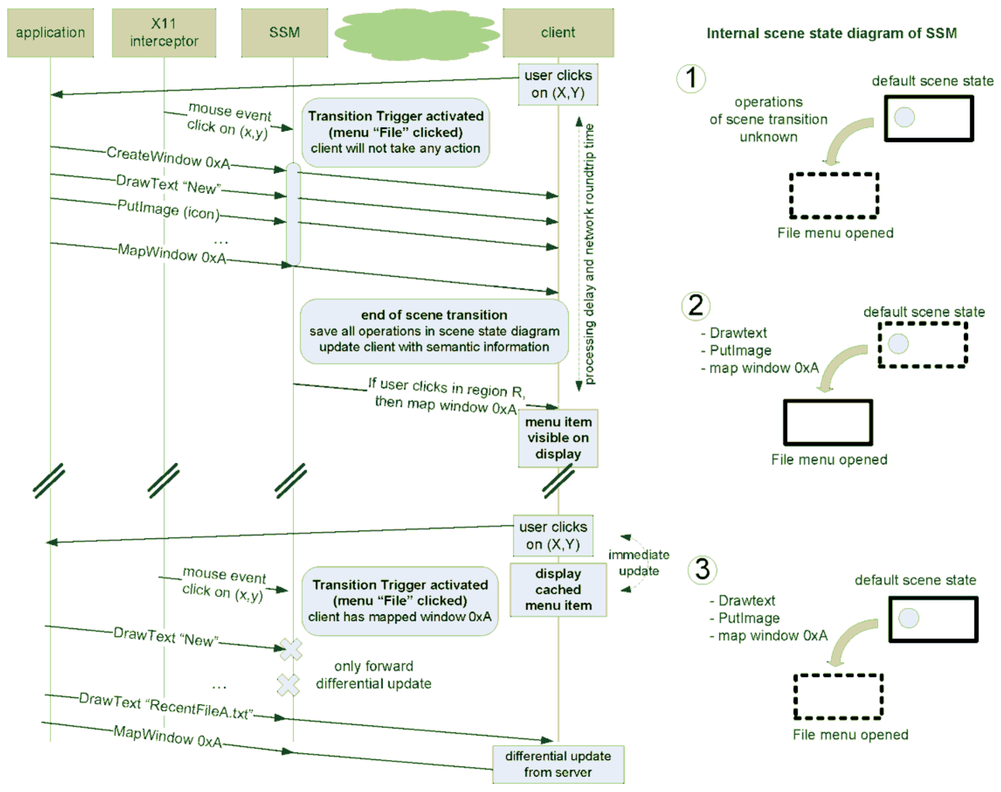
application
X11 interceptor
SSM
client
user clicks on (X,Y)
mouse event click on (x,y)
Transition Trigger activated (menu "File" clicked) client will not take any action
CreateWindow 0xA
DrawText "New"
PutImage (icon)
...
MapWindow 0xA
processing delay and network roundtrip time
end of scene transition save all operations in scene state diagram update client with semantic information
If user clicks in region R, then map window 0xA
menu item visible on display
user clicks on (X,Y)
immediate update
display cached menu item
mouse event click on (x,y)
Transition Trigger activated (menu "File" clicked) client has mapped window 0xA
DrawText "New"
only forward differential update
...
DrawText "RecentFileA.txt"
MapWindow 0xA
differential update from server
Internal scene state diagram of SSM
1
operations of scene transition unknown
default scene state
File menu opened
2
- Drawtext
- PutImage
- map window 0xA
default scene state
File menu opened
3
- Drawtext
- PutImage
- map window 0xA
default scene state
File menu opened

Fig. 5. Sequence diagram of the communication between the server side components when the "File" menu is opened twice during the same session. In order not to overload the figure, the X11toBIFS convertor is omitted from the figure, as well as the delivery of user events through the user event handler to the XServer. The right part of the figure illustrates the current status of the scene state diagram that is maintained by the SSM. The second time the File menu is opened, only a differential update is provided.

When the user clicks on an item in the top menu bar of the text editor, the opened menu needs to be displayed at the client. Although the exact encoding format might differ, non-semantic remote display frameworks would essentially instruct the client to draw a rectangular area and generate some drawing instructions to render the menu items. In a semantic approach, a new window object is created, of which the graphical content is updated by drawing instructions.

In our implementation, we have statically configured the different scene states of the text editor, as well as the objects that are marked as transition triggers. When the application is started, the SSM loads these definitions. These definitions do not contain the actual operations of the scene transitions. As we will discuss below, these actual operations are stored at run-time when they occur. In the future, we foresee to construct the complete scene state diagram at run-time, by dynamically correlating user events to objects in the scene.

When the user clicks the “File” menu item, the SSM detects that a transition is triggered of which the actual operations are not yet available in its scene state diagram. Therefore, all X11 protocol messages are, after conversion to BiFS semantics, forwarded to the client. Consequently, the user experiences both processing and interaction delays between his input and the menu being presented on his screen. At the end of the transition, indicated by the mapping of the menu window on the screen, the application is in a new scene state, and the SSM will take two actions. First, the SSM updates its scene state diagram, storing both semantic operations that create or delete visible objects on the screen, such as operations to map a window object, and drawing operations that change the graphical characteristics of scene objects, *e.g.* putting some text characters in the window object. Second, the SSM will enable the client to handle autonomously the next time the same user event occurs by informing the client on the transition trigger object, *i.e.* the “File“ menu item, and on which semantic operations should be taken, *i.e.* which window should be mapped. A concrete example of how this is translated in BiFS format is shown in Listing 1. A TouchSensor is attached to a rectangular area in the screen, and will call the JavaScript function to make the object visible by setting its scale.

After some time, the user opens the same menu again. This time, the client is able to correlate the user event with a trigger object, and it immediately maps the cached version of the menu window on the screen. To keep both endpoints synchronized, the user event is still transmitted to the application. As the application is unaware of the client caching mechanism, it will generate the same X11 instructions to draw the menu items and map the window on the screen. The SSM will compare these X11 instructions with the information in its scene state diagram. Drawing instructions that will not modify the cached version of the window object, are not forwarded. Only differential drawing instructions are forwarded, for example when the “File“ menu contains a new list of recently opened files. Semantic operations of the scene state transition, such as the mapping instruction, are always forwarded, to ensure the consistency between client and server. Because these semantic operations have already been performed by the client, they not incur an additional visual effect. Because the object was already made visible to the client, the differential updates are immediately shown when they arrive, such as the list with recently opened files. As many menu items and dialog windows do not change, differential updates will only be required in particular cases.

Listing 1: BiFS instructions to execute a JavaScript when the user clicks the mouse on a rectangular item in the scene. The script makes the menu visible by setting the scale argument.

```
DEF TR1 Transform 2D {
  translation Xpos Ypos
  scale  K M
  children [
    Shape {
      appearance Appearance{
        material Material2D{
          emissiveColor R G B
        }
      }
      geometry Rectangle {
        size Height Width \\the menu item
      }
    }
    DEF TS1 TouchSensor {}
    DEF SN1 Script {
      eventIn SFBool MouseClicked
      field SFVec2f ScaleBig 1 1
      field SFVec2f ScaleSmall 0 0
      eventOut SFVec2f CurrentScale
      url “javascript:
        function MouseClicked(value) {
          if(value == true)
            CurrentScale == ScaleBig;
          else
            CurrentScale == ScaleSmall;
        }
        “
    }
  ]
}

ROUTE TS1.isActive TO SN1.MouseClicked
ROUTE SN1.CurrentScale TO TR1.scale
```

6 Experimental evaluation

We have experimentally evaluated the approach of semantic remote display frameworks in two ways. First, it can be expected that the additional transfer of semantic information might result in an increase of the bandwidth consumption. To this end, we compare the bandwidth consumption of our BiFS prototype with conventional, non-semantic architectures. Second, we quantify the benefits of the client side user input handling by measuring the data peaks observed in remote display protocol traffic after some user input.

6.1 Testbed

For the experiments, the testbed presented in Figure 6 was set up. It is composed of a server and client machine, interconnected by an impairment node. This impairment node is equiped with the Click router framework [16] and allows to simulate the propagation delay between client and server.

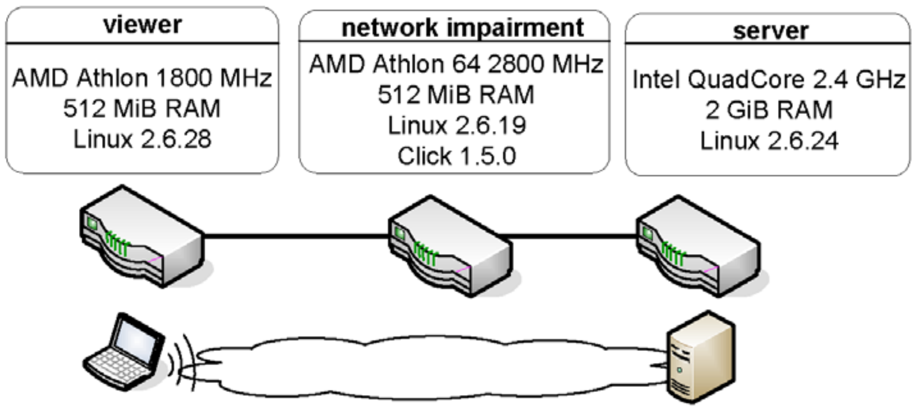

Fig. 6. Structure of the testbed used for the experimental validation.

In order to benchmark the performance of the BiFS prototype, three other open source remote display architectures were installed on the testbed: RealVNC v4.1.1 [27], TightVNC v1.3.10 [37] and FreeNX v3.4.0 [8]. Although all these architectures are Linux based, their bandwidth consumption is in the same order of magnitude of commercially available Windows-based architectures [5,31], such as Microsoft RDP or Citrix XenApp, and consequently they can be considered as a relevant benchmark.

RealVNC is widely used and is based on the Remote Framebuffer protocol which divides the display in multiple rectangles. TightVNC is an enhanced version of RealVNC, offering an additional JPEG-based compression for display updates. It is mainly targetting applications that generate display updates with complex color patterns. FreeNX is an optimized version of the original X architecture. It compresses X11 protocol traffic between the remote application and the rendering XServer running at the client. By using message-specific compression algorithms and extensive client side caching, FreeNX achieves high compression ratios. The bulk of the rendering however is performed by the XServer running at the client. Compared to other remote display technologies, FreeNX requires important memory and CPU resources for cache management, decoding and rendering [6]. Consequently, FreeNX might not be applicable on resource constrained mobile thin client devices.

6.2 Bandwidth consumption of semantic remote display

To assess the bandwidth penalty induced by the additional transfer of semantic scene information, we have selected two scenarios: text editing and web browsing. The text editing scenario covers applications with an elementary scene composition: the display of a text editor is mainly composed of one large and initially blank input field. The display, however, is frequently updated, *i.e.* each time the user enters a character. By contrast, the web browsing scenario covers applications with less frequent display updates, but with a more complex scene composition since web sites typically contain a mixture of text, images, photos and even audiovisual streams.

To simulate the text editing scenario in a realistic way, 5 colleagues were asked to transscribe the same text of 879 characters while recording their user input. Afterwards, these traces were replayed by means of the Xnee tool [30], to ensure that the same user input is used across all tested remote display architectures. The remote application was the gedit text editor [9].

Figure 7 presents the bandwidth consumption, averaged over 10 iterations. For the network roundtrip delay, three different values were configured in the Click router: 0 ms, 30 ms and 100 ms. The particular case of 0 ms was added because some remote display protocols apply different compression mechanisms when the roundtrip time between client and server is limited. In turn, 30 ms is a realistic value of the roundtrip time on wireless links [39, 40], whereas 100 ms is close to the upper limit of network roundtrip times that remain unnoticed to users, as explained in section 2.

Of all tested remote display architectures, TightVNC consumes most bandwidth in the text editing scenario. This can be attributed to the use of JPEG-based encoding, which is optimized for complex, multi-colored graphics, but inappropriate for elementary graphics only containing characters. These results confirm that it is necessary to adapt the compression mechanism to the graphical characteristics of the content. FreeNX is the most bandwidth efficient remote display architecture. As explained above, this is because FreeNX trades network bandwidth for client side processing and requires to run a complete XServer at the client. As a result, FreeNX requires two times as much client CPU and memory compared to BiFS, as can be seen in Table 1. This table compares the resource requirements of the client side application of BiFS (i.e. the

GPAC player) and FreeNX. The incurred load by FreeNX is the sum of the load generated by two processes: the decompression library (*nxssh)* and the XServer (*Xorg)*, whereas the reported load for BiFS is the load of the GPAC player.

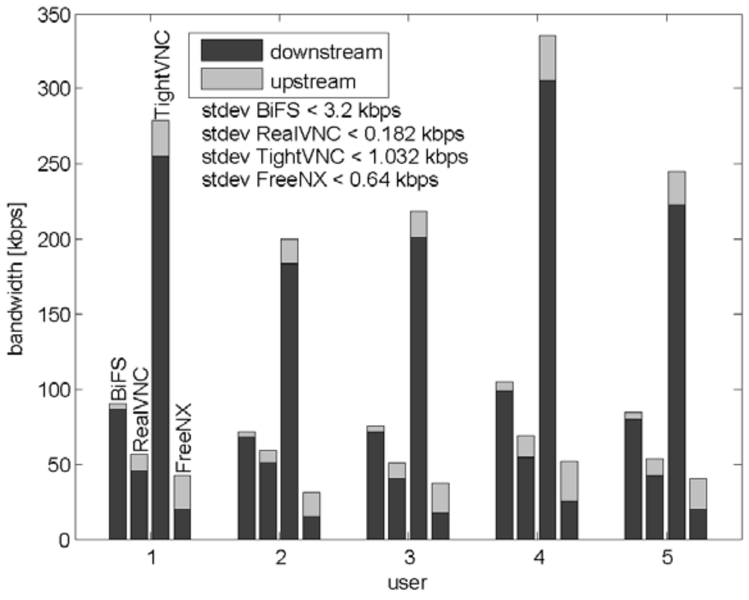

Fig. 7. Average bandwidth consumption for users entering a text of 879 characters, with a roundtrip delay of 30 ms between client and server. The results are the average of 10 simulations per user. BiFS is not optimal for applications with an elementary scene composition. In the upstream direction, BiFS requires significantly less bandwidth, owing to the lack of explicit display update requests (RealVNC and TightVNC) or important client-server synchronization (FreeNX).

Table 1. Comparison of the client side CPU and memory usage of BiFS (GPAC player) and FreeNX for the text editing scenario with a roundtrip delay of 30 ms. The results are averaged over all users, with 10 iterations per user. Measurements were performed on AMD Athlon 1800 MHz with 512 MB.

	memory [MB]	CPU [%]
FreeNX (*nxssh + Xorg)*	59.65	24.9
BiFS	28.75	11.88

Figure 7 shows that BiFS is less optimized for the case of text editing, with elementary graphic updates and scene composition. We believe the BiFS performance can be enhanced by a better transmission strategy. In our current implementation, each BiFS instruction to render a character is transmitted in a separate TCP/IP packet, whereas RealVNC and TightVNC group multiple characters in a single display update. Nevertheless, its bandwidth requirements do not exceed 150 kbps and remain well below the bandwidth offered by modern mobile radio technologies [BALACHANDRAN]. The same conclusions can be drawn for other values of the network roundtrip time. In Table 2 and Table 3, we have presented the results of the same experiment for a network roundtrip time of 0 ms and 100 ms respectively.

Table 2. Average bandwidth consumption for identical experiment conditions as in Fig. 7, but with a roundtrip delay of 0 ms between client and server.

user	BiFS		RealVNC		TightVNC		FreeNX	
	up	down	up	down	up	down	up	down
1	4.48	87.22	12.05	45.67	25.51	261.71	21.58	20.43
2	3.28	67.70	8.84	51.18	18.49	188.63	15.27	14.95
3	3.61	73.20	9.96	40.65	20.13	205.38	17.83	17.59
4	6.79	118.87	19.25	69.96	37.73	339.90	32.55	30.60
5	5.40	96.40	14.72	55.20	30.86	310.11	26.18	25.04

Table 3. Average bandwidth consumption for identical experiment conditions as in Fig. 7, but with a roundtrip delay of 100 ms between client and server.

user	BiFS		RealVNC		TightVNC		FreeNX	
	up	down	up	down	up	down	up	down
1	4.43	81.42	12.15	56.01	20.93	206.08	23.80	21.16
2	3.33	68.23	8.87	51.46	14.40	113.439	17.42	16.27
3	3.58	72.96	10.96	87.14	17.39	145.20	19.89	18.46
4	6.14	113.92	20.60	146.72	28.16	270.28	35.57	31.25
5	5.22	92.47	16.02	25.77	25.34	254.63	28.85	26.10

In the web browsing scenario, four websites were visited: a news website (www.bbc.co.uk), the website of a project on mobile thin clients (www.mobithin.eu), a page on the photo website Flickr and a page search on Google Image. The websites were visited through the Epiphany browser [7] running on the server. User interaction was limited to clicking on one of the four predefined bookmarks in the browser to load the website.

The web browsing scenario was specifically selected to compare the display update compression efficiency of the test remote display frameworks. First, websites have a more complex scene composition compared to the text editing scenario, combining text, images, photos and even audiovisual streams. Second, user interaction is now limited to a single mouse click to load a web page, in contrast to the text editing scenario with frequent user interaction.

Figure 8 shows the total number of bytes exchanged between client and server when loading an individual webpage, allowing to compare the compression efficiency of the evaluated remote display frameworks. Because the number of bytes sent upstream is limited, we provide detailed figures on the bytes sent upstream in Table 3. With BiFS, only one mouse event is sent upstream to communicate the click on the bookmark. TightVNC and RealVNC regularly sent requests for a display update to the server and hence exhibit higher upstream data traffic. With FreeNX, more synchronization traffic between client and server is required.

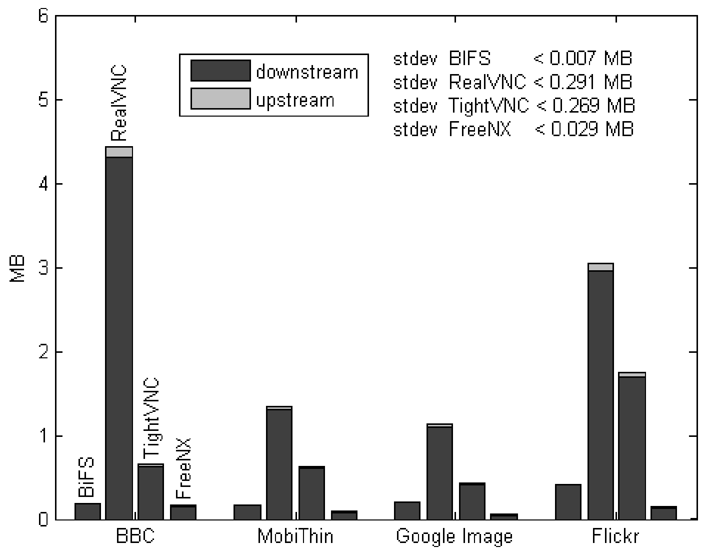

Fig. 8. Total number of bytes exchanged between client and server when loading a website. The results are averaged over 10 iterations. The BiFS compression is significantly better than the compression achieved by RealVNC and TightVNC. Results are for a roundtrip delay of 30 ms.

Table 4. Total number of bytes sent from client to server. These bytes include a single user event (mouse click), requests for display updates (TightVNC and RealVNC) or synchronization information (FreeNX). The results are in kB. Results are for a roundtrip delay of 30 ms.

	BBC	MobiThin	Google Image	Flickr
BiFS	0.4	0.4	0.4	0.4
RealVNC	123.9	42.4	32.23	79.5
TightVNC	32.7	19.5	11.9	41.9
FreeNX	25.2	15.1	12.4	17.8

In terms of the number of bytes required to encode the display updates, BiFS clearly outperforms RealVNC and TightVNC for a roundtrip time of 30 ms. Depending on the visited website, the relative bandwidth reduction varies between 52.0 and 75.4 % compared with TightVNC, and 81.9 % - 95.6 % compared with RealVNC. The large range of these experimental results can be explained by the adaptive encoding mechanism that is applied by VNC, depending on the network roundtrip time. Unlike the text editing scenario, TightVNC is more bandwidth efficient than RealVNC, because its JPEG encoding is better suited for the numerous images on the visited websites.

We have repeated the experiments for network roundtrip delays of 0 ms and 100 ms. We have observed that RealVNC is switching to a higher level of compression for the roundtrip delay of 100 ms. In this case, RealVNC becomes again the most bandwidth efficient compared with TightVNC. Compared with RealVNC, BiFS requires between 21.6 % - 85.7 % less bandwidth to encode the display updates. In turn, FreeNX is able to achieve a compression rate which is 2 or 3 times higher than BiFS; at the expense of running a complete XServer at the client.

The results presented in this section demonstrate the feasibility of using semantic remote display frameworks in a mobile thin client context, in particular for applications that generate a lot of images and have complex scene compositions. The additional bandwidth consumption required for the transfer of semantic information does not outweigh the gains that are achieved by a more efficient encoding. In the next section, we demonstrate how the use of semantic information in remote display frameworks results in additional bandwidth reductions, better traffic shaping and latency reduction.

6.3 Data peak reduction through client side handling of user input

As explained in section 2, the availability of semantic scene information at the client can mitigate the data peaks due to display updates are triggered by user events. To validate this approach, we have created a scene state diagram of the gedit text editor, defining each menu item and icon in the top menu bar as a transition trigger. Two different types of scene transitions were considered. In the first experiment, the File menu was opened 3 times in the same session by clicking on a menu item, whereas in the second experiment, the Save dialog window is called by clicking on the corresponding icon. As was illustrated in Figure 5, the semantic information on the transition trigger and the corresponding scene transition is transferred after the menu or dialog window have been opened for the first time. From now on, the client can immediately show these objects when a similar user event occurs. No server side instructions are required, except for differential updates or to draw a tooltip. Consequently, the interaction latency is reduced to the time required for the local processing at the client.

Figure 9 shows the instantaneous bandwidth consumption when the File menu is three times opened during the same text editing session. When the user clicks the menu for the first time, the menu is not yet available in the client cache. The server instructs the client to save this menu and provides the client the required semantic scene information to directly show the cached object the next time. This can be observed in the figure when the user opens the menu for the second and the third time: only very limited downstream traffic is observed.

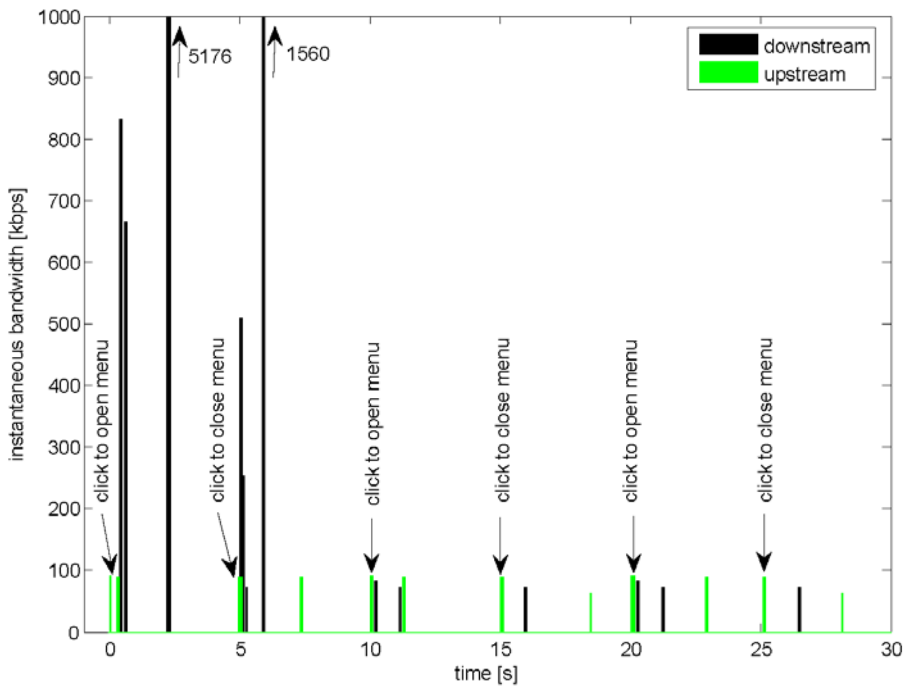

Fig. 9. Instantaneous BiFS bandwidth consumption, measured during a sample experiment in which the same menu was opened and closed three times. The second and third time, a significant reduction of the peaks following the mouse click can be observed.

Table 5 provides quantitative results on the peak reduction by comparing the total number of bytes that is sent from server to client when the menu or dialog window are shown. It is observed that all solutions sent less data the second time the user operation occurred. For TightVNC and RealVNC, this peak reduction can be mainly attributed to the zlib compression efficiency. This technique employs a dictionary with recently encoded byte strings. Because the reported figures are generated when the same menu is consecutively opened without any other operation in between, the zlib compression is able to reduce the peaks by 18.1-30.5 % for the menu and by 20.5-25.2 % for the dialog window. In regular sessions, other display updates will be transmitted to the client in between two menu openings and the zlib compression efficiency will decrease.

Table 5. Total number of bytes sent from server to client when repeatedly opening the File menu or the Save dialog of the gEdit application in the same session. The presented results are the average of 10 iterations and were obtained with a network roundtrip time value of 30 ms. The addition of the SSM results in an additional data reduction of 22.1-23.1 %.}

	open menu		reduction	open dialog window		reduction
	1st time	2nd time	1→2 [%]	1st time	2nd time	1→2 [%]
BiFS w/o SSM	22 735	5 634	75.2	23 190	6 705	71.1
BiFS with SSM	23 407	417	98.2	23 789	1 628	93.2
FreeNX	6 302	2 267	64.0	12 533	8 624	31.2
TightVNC	16 423	11 419	30.5	14 037	11 162	20.5
RealVNC	241 332	197598	18.1	300 270	224 459	25.2

The data peak reduction achieved by FreeNX is caused by two effects. First, FreeNX achieves a high compression ratio due to caching of X11 messages at the client. Second, when a menu is opened for the first time, additional X11 instructions are required to create and configure a new window object, besides the drawing instructions to render the content of the window. When the user opens the menu or dialog window for the second time, less X11 instructions are required because the previous window is made visible again, instead of creating a new window object. Because BiFS directly translates each X11 protocol message, this second effect is also the reason why BiFS is able to reduce the amount of data to be sent by 71.2-75.2 %, even when no Scene State Manager is implemented. The addition of the Scene State Manager results in additional reduction of 22.1-23.0 %. By comparing the data that is sent the first time a user event occurs, it can be concluded that less than 1kB is required to transfer the additional semantic information to allow client side user input handling.

7 Conclusion

Semantic remote display architectures enable the client to identify the individual objects in the displayed scene by providing information on the composition of the scene and the individual characteristics of each object. In this paper, we have demonstrated how semantic information can be exploited at the client to react immediately to user input, instead of having to wait for the display updates of the server. At the server, a Scene State Manager captures semantic information from the application and informs the client how the user can manipulate the objects in the displayed scene. Furthermore, we have detailed the implementation details of our prototype that uses MPEG-4 BiFS to encode and transfer the semantic information to the client. We have experimentally assessed the additional bandwidth that is required to convey semantic information to the client. Lastly, we have demonstrated how handling user events at the client mitigates the data peaks of remote display protocol traffic by on average 23 %.

In our current implementation, the different scene states and the transition triggers are loaded from a preconfigured file. Future research will focus on the detection of the individual scene states at run-time, by correlating the user input to the resulting display update. This should result in application profiles that can be reused for other users that use the same application. Furthermore, we will concentrate our efforts on optimizing the amount of semantic information that needs to be saved in the memory of the resource-constrained thin client device. Currently, each scene transition is saved at the client after its first occurrence. This can be optimized by creating individual user profiles, indicating the most relevant objects to be cached at the client, *e.g.* the menus and dialog windows that are most frequently visited. Lastly, we will add more types of client side actions. Currently, the client is only enabled to display windows in response to a mouse click. The range of actions can be extended with, for example, the client directly contacting the source of audio and video streams.

8 References

[1] Balachandran K, Bi Q, Rudrapatna, A, Seymour J, Soni R, Weber A (2009): Performance Assessment of Next-Generation Wireless Mobile Systems. BELL LABS TECHNICAL JOURNAL 13(4): 35-58

[2] Boukerche A, Pazzi RWN, Feng J (2008) An end-to-end virtual environment streaming technique for thin mobile devices over heterogeneous networks. COMPUTER COMMUNICATIONS 31(11): 2716-2725

[3] Citrix Systems Inc.: http://www.citrix.com. Last retrieved on 2011-04-20.

[4] Claypool M, Claypool K (2006) Latency and player actions in online games. COMMUNICATIONS OF THE ACM 49(11):40-45

[5] Descampe A, De Vleeschouwer C, Iregui M, Macq B, Marques F (2007) Prefetching and caching strategies for remote and interactive browsing of JPEG-2000 images. IEEE TRANSACTIONS ON IMAGE PROCESSING 16(5):1339-1354

[6] Deboosere L, De Wachter J, Simoens P, De Turck F, Dhoedt B, Demeester P (2007) Thin client computing solutions in low- and high-motion scenarios. In proceedings of the Third International Conference on Networking and Services (ICNS 2007), pp. 230-235

[7] Epiphany, the official web browser of the GNOME desktop environment: http://projects.gnome.org/epiphany/ Last retrieved on 2011-04-20

[8] FreeNX. Http://freenx.berlios.de Last retrieved on 2011-04-20

[9] gEdit, the official text editor of the GNOME desktop environment: http://projects.gnome.org/gedit/ Last retrieved on 2011-04-20

[10] Hua ZG, Xie X, Liu H, Lu HQ, Ma WY (2006) Design and performance studies of an adaptive scheme for serving dynamic Web content in a mobile computing environment. IEEE TRANSACTIONS ON MOBILE COMPUTING 5(12):1650-1662

[11] ISO/IEC (2005) Coding of audio-visual objects - part 11: Scene description and application engine. ISO/IEC 14496-11

[12] Joveski B, Mitrea M, Preteux F (2010) MPEG-4 LASeR-based thin client remote viewer. In Proceedings of 2nd European Workshop on Visual Information Processing (EUVIP), Paris, FRANCE.

[13] Joveski B, Simoens P, Gardenghi L, Marshall J, Mitrea M, Vankeirsbilck B, Preteux F, Dhoedt B (2011) Towards a multimedia remote viewer for mobile clients. In Proceedings of the Multimedia on Mobile Devices and Multimedia Content Access, SPIE, San Francisco, USA.

[14] Khin HS, Kim S (2007) An analyzer of the user event for interactive DMB. In proceedings of Third International Conference on Embedded Software and Systems (ICESS 2007). Lecture Notes in Computer Science (vol.4523), pp. 818-25

[15] Kim HC, Leong, JM, Kim K (2004) Development of interactive contents streaming system based on mpeg-4. In proceedings of the 6th International Conference on Advanced Communication Technology, vol. 2, pp. 751-755

[16] Kohler E, Morris R, Chen B, Jannotti J, Kaashoek M (2000) The Click modular router. ACM TRANSACTIONS ON COMPUTER SYSTEMS 18(3), 263-297

[17] Koller D, Turitzin M, Levoy M, Tarini M, Croccia G, Cignoni P, Scopigno R (2004) Protected interactive 3D graphics via remote rendering. ACM TRANSACTIONS ON GRAPHICS 23(3):695-703

[18] ISO/IEC (2008). Coding of audio-visual objects – Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)

[19] Jansen J, Bulterman DCA (2009) SMIL State: an architecture and implementation for adaptive time-based web applications. MULTIMEDIA TOOLS AND APPLICATIONS 43(3):203-224

[20] Le Feuvre J, Concolato C, Moissinac JC (2007) Gpac: open source multimedia framework. In Proceedings of the 15th International Conference on Multimedia (MULTIMEDIA '07), pp. 1009-1012

[21] Lethanhman C, Isokawa H, Kato T (2009) Multipath data transmission for wireless thin clients. In Proceedings of the 3rd International Conference on Mobile Ubiquitous Computing Systems, Services and Technologies (UBICOMM), Malta.

[22] Live555: Live555 streaming media. http://www.live555.com Last retrieved on 2011-04-20

[23] MicroSoft: Remote desktop protocol: Basic connectivity and graphics remoting specification. http://msdn.microsoft.com/en-us/library/cc240445 Last retrieved on 2011-04-20

[24] Mitrea M, Simoens P, Joveski B, Marshall J, Taguengayte A, Prêteux F, Dhoedt B (2009) BiFS-based approaches to remote display for mobile thin clients. In Proceedings of the SPIE - The International Society for Optical Engineering, vol. 7444, p. 74440F (8pp.)

[25] Paravati G, Celozzi C, Sanna A, Lamberti F (2010) A Feedback-Based Control Technique for Interactive Live Streaming Systems to Mobile Devices. IEEE TRANSACTIONS ON CONSUMER ELECTRONICS 56(1): 190-197

[26] Pendyala VS, Shim SSY (2009) The web as the ubiquitous computer. COMPUTER 42(9):90-92

[27] Preda M, Villegas P, Moran F, Lafruit G, Berretty RP (2007) A model for adapting 3D graphics based on scalable coding, real-time simplification and remote rendering. VISUAL COMPUTER 24(10):881-888. International Conference on Cyberworlds, Hannover, GERMANY, OCT 24-27, 2007

[28] RealVNC: Virtual network computing. http://www.realvnc.com Last retrieved on 2011-04-20

[29] Richardson T, Staford-Fraser Q, Wood K, Hopper A (1998) Virtual network computing. IEEE INTERNET COMPUTING 2(1):33-38

[30] Sandklef H (2004) Testing applications with Xnee. LINUX JOURNAL 117:87-90

[31] Schlosser D, Binzenhoefer A, Staehle B (2007) Performance Comparison of Windows-based Thin-Client Architectures. In: 2007 AUSTRALASIANTELECOMMUNICATION

NETWORKS AND APPLICATIONS CONFERENCE, pp. 223-228. Australasian Telecommunication Networks and Applications Conference, Christchurch, NEW ZEALAND, DEC 02-05, 2007

[32] Schlosser D, Staehle B, Binzenhofer A, Boder B (2010) Improving the QoE of Citrix Thin Client Users. In: ICC 2010 - 2010 IEEE International Conference on Communications

[33] Signes J, Fisher Y, Eleftheriadis A (2000) MPEG-4's binary format for scene description. SIGNAL PROCESSING-IMAGE COMMUNICATION 15(4-5):321-345

[34] Simoens P, Praet P, Vankeirsbilck B, De Wachter J, Deboosere L, De Turck F, Dhoedt B, Demeester P (2008) Design and implementation of a hybrid remote display protocol to optimize multimedia experience on thin client devices. In: ATNAC: 2008 AUSTRALASIAN TELECOMMUNICATION NETWORKS AND APPLICATIONS CONFERENCE, pp. 391-396. Australasian Networks and Applications Conference 2008, Adelaide, AUSTRALIA, DEC 07-10, 2008

[35] Sun Y, Tay TT (2008) Analysis and reduction of data spikes in thin client computing. JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 68(11):1463-1472

[36] Tan KJ, Gong JW, Wu BT, Chang DC, Li HY, Hsiao YM, Chen YC, Lo SW, Chu YS, Guo JI (2010) A remote thin client system for real time multimedia streaming over VNC. In: 2010 IEEE International Conference on Multimedia and Expo (ICME), pp. 992-997

[37] TightVNC: Tightvnc. Http://www.tightvnc.com Last retrieved on 2011-04-20

[38] Tolia N, Andersen D, Satyanarayanan M (2006) Quantifying interactive user experience on thin clients. COMPUTER 39(3):46+

[39] Wang F, Ghosh A, Sankaran C, Fleming PJ, Hsieh F, Benes SJ (2008) Mobile WiMAX systems: Performance and evolution. IEEE COMMUNICATIONS MAGAZINE 46(10):41-49

[40] Wellnitz O, Wolf L (2010) On latency in IEEE 802.11-based wireless ad-hoc networks. In 2010 5th International Symposium on Wireless Pervasive Computing (ISWPC), pp. 261-266