On the Support for Heterogeneity in Networked Virtual Environment

Hiroshi Fujinoki

Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL 62026-1656, USA
+1 618 650 3727

hfujino@siue.edu

ABSTRACT

This paper presents our ongoing research activity to design and implement a framework for an networked virtual environment (NVE) that efficiently supports both hardware and software heterogeneity. In the proposed framework, three new techniques, application layer multicast transmission-rate pruning, fairness control for delay-sensitive activities (token-bucket algorithm) and bandwidth compensation by a combination of server-side and client-side dead reckoning, are designed, proposed and integrated in the new framework that supports heterogeneous networks and end systems.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *Network communications, Network topology*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Client/server, Distributed applications*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism – *Virtual reality*

General Terms

Design, Performance

Keywords

Networked virtual environments, heterogeneous networks, event synchronization, multicast

1. INTRODUCTION

Networked virtual environments (NVEs) are network applications where multiple users at networked computers participate in shared virtual real-time activities, such as online games, simulations and interactive teleconferencing. NVEs are integrated systems in a sense that they gather input from multiple remote users, compile remote users' inputs into common state information (called "the state information" hereafter), render graphic images based on the state information, transmit the graphic images to represent a common real-time virtual environment, and synchronize event inputs by multiple users with the ultimate goal of providing an environment that looks as if it were a local environment.

The essential problem in most NVEs is a trade-off between responsiveness and consistency [1]. Improving consistency is possible by lowering responsiveness, while improving responsiveness usually results in lowering consistency. This trade-off is the primary cause of the difficulty in satisfying the requirement parameters [2]. It is the responsibility of an NVE to control this trade-off under the given environmental factors of available transmission bandwidth, end-to-end delay and local resources at end hosts, to meet the given requirements for consistency, responsibility and scalability. What complicates this problem further is heterogeneity in end-to-end delay, transmission bandwidth and local resources at each client host.

NVEs have recently been getting wide popularity from public Internet users and will be one of the key network applications in the future Internet. The wide popularity of NVEs among general Internet users implies a need for efficient support for the heterogeneous environments. Heterogeneity means use of different hardware and software components in the network infrastructure and local end hosts. For example, heterogeneity in network infrastructure means different types of networks, such as packet switching and circuit switching networks, as well as different transmission methods, such as wireless transmissions, satellite transmissions, wired subscriber loops and direct network connections through LANs. Heterogeneity in local end hosts means differences in available resources and capabilities at a local host computer.

Although there has not been common agreement for what the essential core activities in NVEs are, the following five scenarios have been developed as our foundations of analysis for how various NVE activities can be supported in heterogeneous environments.

1. A hot potato scenario, in which each player takes a turn claiming ownership of a graphical object and then transfers that ownership by means of a virtual toss to another player.
2. A competitive race-condition scenario, in which two or more players simultaneously attempt to seize a single graphical object, but over which only one of them may claim ownership.
3. A cooperative scenario, in which multiple players simultaneously attach themselves to a single graphical object and are then influenced by the virtual forces exerted upon that object by their fellow "owners".
4. A vehicular scenario, in which players enter and exit a conveyance, but only at designated sites at which the conveyance comes to a stop.

5. A tour guide scenario, in which players virtually attach themselves to a specifically designated guide player, whose point of view they perceive until they detach themselves later.

The rest of this paper is organized as follows. Section 2 describes our new NVE framework proposed to efficiently support heterogeneous environments. The three new techniques of application layer multicast transmission-rate pruning, fairness control for delay-sensitive activities (token-bucket algorithm) and bandwidth compensation by a combination of server-side and client-side dead reckoning are described. Section 3 summarizes the conclusions in this study and future work, followed by acknowledgement and a list of the major references.

2. PROPOSED NVE FRAMEWORK

2.1 Dynamic transmission-rate pruning by application layer multicasting

In this section, a new mechanism for dynamic on-demand pruning that adjusts the transmission rate at intermediate multicast routers is introduced. The proposed dynamic transmission-rate pruning is a distributed method in a sense that it does not require a single host (i.e., the server) to perform the pruning. The proposed mechanism dynamically prunes transmission-rate, based on the level of details (LODs) specified by each of the terminal receivers in a heterogeneous NVE.

A problem in realizing such dynamic transmission-rate pruning is that pruning can not be performed by simply dropping some of the packets that carry image frames from the server. Regarding this point, Funkhouser proposed an adaptive image rendering algorithm to achieve a uniform frame rate by adjusting LODs delivered to clients with different local resources [3], but it does not allow intermediate routers adjust the transmission rate.

We designed the incremental quality image encoding that facilitates application-level multicast pruning at intermediate routers for dynamically adjusting the transmission rate. The incremental quality image encoding involves a definition of data structure that allows intermediate multicast routers to dynamically adjust the data rate while image frames are dropped in a controlled manner for a given target transmission rate. We developed a protocol specification for the application layer multicast pruning that allows intermediate multicast routers to recognize the requirements from each of the heterogeneous clients and networks in its downstream.

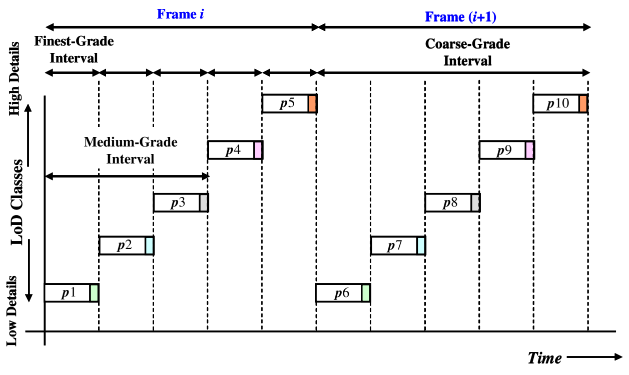

Figure 1. Concept of the incremental quality image encoding for application layer transmission-rate pruning.

For multicast routers to dynamically prune the downlink data rate, image frames at a server are encoded in such a way that each of the image frames is decomposed into multiple classes of LODs and each multicast router can drop the packets that belong to unnecessary classes of LODs. Reduced data rate and reduced details in the transmitted image frames will allow receiving host computers to adjust packet inter-arrival time to give themselves enough time to digest the delivered image information. Figure 1 visualizes the concept of the incremental quality image encoding.

In Figure 1, the y-axis indicates the level of image quality represented by different levels of LOD classes (the higher in the y-axis, the more details are encoded). UDP packets that belong to a high LOD class contain rich details in an image frame while those belonging to a lower LOD class contain fundamental information for an image frame. Thus receiving image data up to the highest LOD class results in the best image quality at the client side, while receiving data of the lowest LOD class results in the minimum quality image.

In Figure 1, packets ***p***1 and ***p***6 are UDP packets that belong to the lowest LOD class, while ***p***5 and ***p***10 are those for the highest LOD class. Note that the lower the LOD class packets belong to, the more essential image information they are carrying. A client that only has enough link bandwidth to receive the lowest quality image should receive only the packets that belong to the lowest LOD class (e.g., ***p***1 and ***p***6) and multicast routers selectively drop packets that belong to any other LOD classes. Similarly, a client that does not have enough link bandwidth to receive the full-quality image should receive the packets for up to an appropriate LOD class.

The packet inter-arrival time will be the shortest for a session that requires the highest quality image (shown as "Finest-Grade Interval" in the figure), while it is longest for the lowest quality session, which needs only the lowest LOD packets (shown as "Coarse-Grade Interval"). As a result, fine-grain control for resultant required data rate can be performed by specifying LOD classes each host would like to tune in. Since each intermediate multicast router performs the pruning, the proposed application-level pruning is distributed and scalable.

The information needed for multicast routers to perform dynamic pruning are multicast address and port number, frame sequence number, LOD-class ID, and fragmentation offset within a LOD class. The first two fields (multicast address and port number) are those already implemented in existing UDP and IP, while the other fields need to be implemented in the application layer. The format of the proposed data structure is shown in Figure 2.

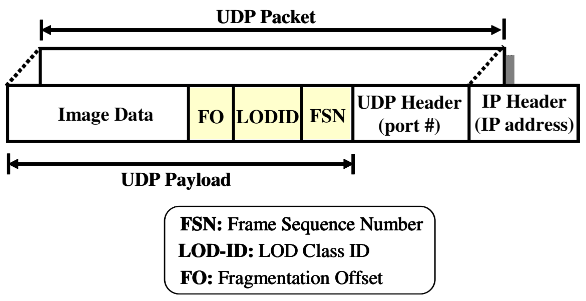

Figure 2. Packet format for the application-layer pruning for transmission rate.

Excess quality traffic for a given client can be filtered at any multicast router by referencing the required LODs. In addition to the transmission rate of the bottleneck link, any of the following factors can be used to filter excess traffic:

- Processing horsepower at a client host
- Network congestion
- Complexity in displayed images

2.2 Fairness control for delay-sensitive NVE applications

This section describes a new event synchronization algorithm, the token bucket algorithm, which is designed to improve the fairness for heterogeneous clients in updating the state information at the server. The algorithm also eliminates the need for the global clock. When a server renders an image frame, the server randomly generates a token, attaches the token to each new frame and transmits a frame to clients. When a client receives an image frame, it extracts a token, and holds the token until the player makes a decision for the next activity. When the client sends its next activity to the server, the client attaches the token to the message to the server.

Figure 3 illustrates the procedure in the token bucket synchronization algorithm. The server accesses the state information, renders an image frame and generates a token for this frame (①). Each token is a randomly generated integer that can not be predicted by clients. The new image frame is then transmitted to all clients with the token. The network protocol processes the data in an image frame into packets and those packets are transmitted at the server (②). The packets are transferred through a network as shown by ③ (one-way end-to-end delay). The client side network protocol receives and processes packets for an image frame from the server (④). A client keeps the token attached to this image frame. The graphic sub-system displays the transmitted image frame at a client (⑤). Human-interaction starts and a reactive input will be made at a client host (⑥). A player sees an image frame, recognizes the semantics, and makes a decision for the next action (⑥-(a)). A human player makes an input for the decided action (⑥-(b)). Application process constructs a message to transmit the player's activity (⑥-(c)). The client attaches the token to this message. The network protocol constructs packets to transmit this message and they are transmitted (⑦). Another one-way end-to-end delay from a client to a server happens (⑧). The server receives and processes the client's message (⑨). The state information at the server is updated (⑩) and the procedure repeats from ①.

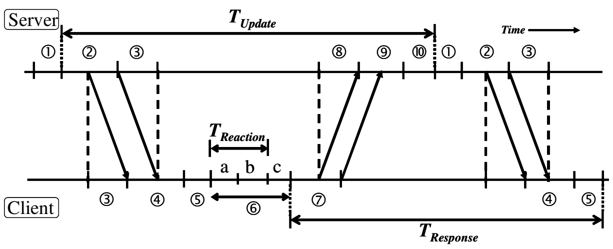

Figure 3. Dissection of latency in user interaction.

In Figure 3, the player's response time ($\boldsymbol{T_{Response}}$) is defined as the time since a player inputs its activity until the player sees the update from the server. The IEEE's DIS standard recommends that the upper threshold for the response time be 200ms [4]. The update time ($\boldsymbol{T_{Update}}$) is the time interval between two image frames that contain updates added by the server to reflect players' inputs and it is called an *update window*. The time available for a player to respond to meet this update window is from the beginning of (⑥-(a)) until the end of (⑥-(b)). This time slot is called a *reaction window* ($\boldsymbol{T_{Reaction}}$). Any input that is made after the reaction window will still be transmitted to the server, and will be used to update the state information in the next update window.

The length of the update window can be adjusted depending on such factors as dynamic fluctuations in end-to-end delay and in transmission bandwidth possibly due to network congestion, and the membership dynamically changes. The interval should also be adjusted based on the requirements on responsiveness for the activities performed in an NVE.

The token bucket algorithm has another parameter, the update margin ($\boldsymbol{U_{Margin}}$), which indicates the number of update window slots a late message can miss. For example, if $\boldsymbol{U_{Margin}} = 1$, late messages will be used to update the state information if they are delivered to the server by the end of the next update window after its expected window. If $\boldsymbol{U_{Margin}} = 2$, messages delivered to the server by the end of the third update window (including its expected window) will be accepted, while any late message will be discarded, if $\boldsymbol{U_{Margin}} = 0$.

On receiving a message from a client, the server performs one of the following procedures:

(a) If the token is the one expected for the current update window, the server checks if this is the first message from this client for this update window. If it is, the server holds this message. If not, the server replaces the old message by this message. At the end of the current update window (⑩), the server updates the state information using the latest message from this client.

(b) If the token is for one of the previous update windows within the last $\boldsymbol{U_{Margin}}$ slots, the server checks if there *was* any message from this client that already updated the state information in that previous update window. If there was, this late message is dropped (this is for preventing the server from accepting cheating messages created using one of the previous tokens). If not, the state information is immediately updated using this late message.

(c) If the token does not match any of the previous update windows (the server needs to remember the tokens for up to $\boldsymbol{U_{Margin}}$ previous update windows), the message is either too late or illegal and the message is dropped immediately.

2.3 Bandwidth compensation by a combination of server-side and client-side dead reckoning

The upper threshold for the delay between a player input and its playout is 200ms while an image refresh rate of 25fps (40ms interval) will give a smooth playout [4, 5]. These statistics imply that it will be enough if only one out of several image frames (at least one in every five frames for the above case) created by the server reflects updates by human players. Based on this analysis, two different types of frames are defined. *The update frames* are those generated at the server to reflect players' inputs and they transmit the image frames that contain the new state information updated based on players' inputs. *The gap frames* are those created to provide a smooth transition between two update frames.

Dead reckoning was initially proposed to save transmission bandwidth by letting clients locally generate image frames. Dead reckoning predicts motion paths in other players' activities and changes in background, generates image frames, and performs convergence if the prediction is incorrect [2, 6]. In a situation where players are actively moving, smoothness can be improved by locally predicting the players' activities at a client under limited transmission bandwidth and long end-to-end delay. This mechanism is called *client-side dead reckoning*.

A new scheme for dead reckoning, *server-side dead reckoning*, which complements client-side dead reckoning in heterogeneity support, is proposed. The server-side dead reckoning predicts motions of remote players, and generates gap frames. Server-side dead reckoning combined with the fairness control by the token bucket synchronization algorithm is visualized in Figure 4. After the end of ②, the server-side dead reckoning starts, generates gap frames and transmits them to clients at a fixed interval of Δ. The frame rate for the gap frames should be any length less than 40ms (assuming the target of 25fps) or it can be up to half of T_{Update}, if a client does not have enough local resources or network transmission bandwidth as long as the requirements for a NVE application permit.

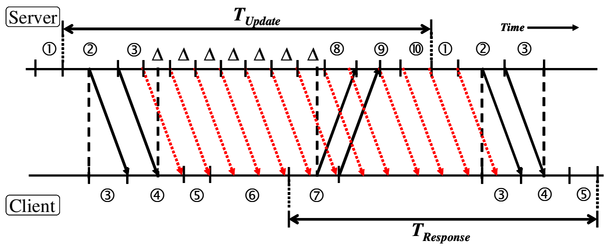

Figure 4. Visualized server-side dead reckoning combined with the token bucket algorithm

The server-side dead reckoning is combined with the application layer multicast transmission-rate pruning to support heterogeneous NVEs. The frame interval and the LOD in the server-side dead reckoning can be adjusted by considering the network and clients' local resources. For example, if the available transmission bandwidth is high and the local processing resource is low, the client should take full advantage of the server-side dead reckoning. The server generates intermediate frames with rich details and what a client has to do is just to receive the synthesized image frames from the server and locally display them without much local processing workload. The other extreme is if the available transmission bandwidth is low but the local resource, such as graphics and processing resource, is abundant; in the cases the dead reckoning can be performed in the client side.

In case of client-side dead reckoning, the server still needs to transmit essential information to clients to allow them to perform client-side dead reckoning. The amount of information that still needs to be transferred to a client depends on the LOD requirement at the client and can be controlled by the dynamic multicast pruning proposed before.

3. CONCLUSIUONS AND FUTURE WORK

It is expected that networked virtual environments will be major applications in the near-future Internet. We also expect that the diffusion of the Internet technologies to general Internet users will increase the need for supporting heterogeneous environments in both the network infrastructure and the capabilities at end hosts. As a result from our survey, we found that existing solutions for the three key implementation components, communication models, data transmission methods and event synchronization algorithms, are not well designed for flexibly supporting various virtual-reality activities in NVEs and for supporting heterogeneous networks and end hosts.

We designed and propose a new NVE framework to support a wide variety of activities in heterogeneous networked environments. To improve heterogeneity support, we designed three new techniques of application layer multicast transmission-rate pruning, fairness control for delay-sensitive activities (the token-bucket algorithm) and bandwidth compensation by a combination of server-side and client-side dead reckoning.

We plan to conduct performance studies by building a prototype of the new framework to quantify and compare the performance of the new framework to existing methods. We are planning to measure the effect of heterogeneous support by measuring the frame drop rate, network traffic scalability by measuring the network traffic load actually generated in a network, responsiveness by measuring the average response time for clients with different local resources and fairness by interviewing human users after they try NVEs implemented by the proposed framework and other working systems.

4. ACKNOWLEDGEMENT

The five NVE scenarios were designed by Dr. William White. This project is the results of inspirations from our discussions.

5. REFERENCES

[1] Smed, J., Kaukoranta, T., and Hakonen, H. *A Review on Networking and Multiplayer Computer Games*. Technical Report 454, University of Turku Centre for Computer Science, Turku, Finland, 2002.

[2] Singhal, S. and Zyda, M. *Networked Virtual Environments: Design and Implementation*. Addison Wesley, Reading, MA, 1999.

[3] Funkhouser, T., and Sequin, C. Adaptive Display Algorithm for Interactive Frame Rates during Visualization of Complex Virtual Environment. *Computer Graphics*, 27 (1993), 247-254.

[4] *IEEE Standard for Distributed Interactive Simulation - Communication Services and Profiles*. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[5] Berglund, E. J., and Cheriton, D. R. Amaze: A Multiplayer Computer Game. *IEEE Software*, 2, 1 (May 1985), 30-39.

[6] Gautier, L., Diot, C., and Kurose, J. End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet. In *Proceedings of 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)* (New York, NY, March 21-25, 1999), IEEE Computer Society Press, Los Alamitos, CA, 1999, 1470-1479.