Robust Transmission of 3D Geometry over Lossy Networks

Zhihua Chen
Dept. of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
zhihua.chen@vanderbilt.edu

Bobby Bodenheimer
Dept. of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
bobbyb@vuse.vanderbilt.edu

J. Fritz Barnes
Dept. of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
J.Fritz.Barnes@vanderbilt.edu

ABSTRACT

This paper describes a robust mechanism for transmitting 3D meshes over the Internet. TCP/IP is an excellent means for reliable transport over the Internet. However, multi-user, real-time graphics applications may find TCP transmission disadvantageous when reception of a mesh is time-critical. To improve speed one could use an unreliable transmission protocol. Yet typical mesh compression schemes increase the fragility of the mesh to lossy transmission. In this paper, we develop a hybrid method of transmitting meshes over the Internet, built upon progressive mesh [17] technology. The hybrid method transmits important visual detail in a lossless manner, but trades off loss of visually less important detail for transmission speed. Tests of the method in a lossy network environment show that the method improves the transmission time of the mesh with little degradation in quality.

1. INTRODUCTION

As the Internet expands, demand is growing for high quality 3D geometry in applications, from games such as Everquest [12] to collaborative virtual environments to multimedia and purely web-based applications. Such applications use high resolution 3D meshes to achieve their effect, and the challenge of the growing demand for these meshes is how to store and transmit the large amount of data contained in them. There are two traditional methods for obtaining high resolution 3D meshes for these applications: (1) assume that in a traditional client-server model the client already has the model; (2) wait for the model to be downloaded over the Internet.

Both methods have drawbacks. The traditional client-server model assumes all models are available locally, and therefore is appropriate for such applications as games where models never change, but it is less appropriate for distributed applications where new models may be created by users and incorporated into the environment in real-time. Likewise, downloading may be appropriate in some scenarios, but even if the data is compressed it can take an unacceptably long time to receive a complex model. For example, online video gaming typically uses the first method in obtaining its 3D geometry, because smooth navigation and increased user satisfaction result when there are no delays in rendering caused by not having the model.

A partial solution to these drawbacks is that of Hoppe and others, to progressively transmit the geometric data [17]. In this method, the server initially sends coarse shape information to a client that can be reconstructed and rendered very quickly. Then increasing detail in the model is transmitted to the client, allowing the client to progressively refine the initial model into the full resolution model. If, for example, the object is initially far away, then a user cannot perceive the loss of detail in the model caused by rendering the initial model. More generally, the user is able to see and interact with the coarse model immediately, and thus the delay while the model is being refined is not as perceptually disturbing as delay in the action while the model is fully downloaded. In its basic form, progressive transmission does not *reduce* the amount of data that needs to be transmitted, but it *orders* the data from most important to least important. The crux of our method lies in this ordering: loss of data that is less visually important may be tolerable if the model is transmitted faster.

Various authors [23, 32, 20] have combined mesh compression techniques with progressive transmission to reduce the amount of data that needs to be sent. Typically, there is a tradeoff between compression ratio and the *fragility* of the compressed mesh. For example, if even one packet is lost in a highly compressed mesh then the entire geometry may not be reconstructable from what remains. This fragility mandates the use of a *reliable* transmission method, and the common protocol for such transmission is TCP, which is generally error-free. However, using TCP can result in non-linear delays when transmitting data over lossy network environments. Alternative transmission protocols such as UDP are unacceptable because they may not deliver all packets to the receiver. In many collaborative virtual environments [22, 14], a single sender sends data to multiple recipients over the Internet Multicast Backbone (MBONE) [8]. This technique allows a host to scalably transfer information to multiple recipients. However, multicast communication often does not support reliable communication because of the complex and prohibitive cost associated with developing reliable multicast transport.

In this paper, we are not addressing the issue of mesh compression. Rather, we develop the idea of hybrid transmission: reliable where it is needed, but allowing packet loss where it can be tolerated. In particular, we explore a hybrid scheme for combining TCP and UDP transmission modes, exploiting the property of progressive meshes that important detail is transmitted first, and the loss of some packets may be tolerable if the overall quality of the resulting mesh is good.

When packets are lost, geometry is lost. Our method attempts to trade off the possibility of improving the mesh with new refinements as additional information arrives versus the possibility of

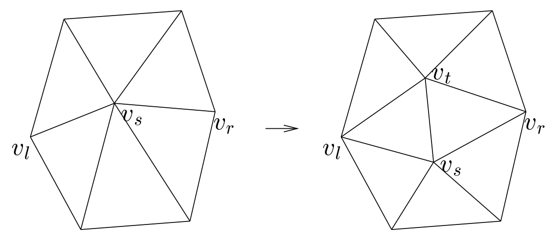

Figure 1: Vertex split transformation.

causing some amount of mesh corruption. This tradeoff can be regulated. Additionally, our method could be applied in conjunction with mesh compression techniques to further reduce bandwidth. Thus, we only consider the basic progressive mesh (PM) scheme of Hoppe [17], as more advanced schemes would typically have similar results in different ratios. We demonstrate on an actual congested network that our scheme outperforms a pure TCP transmission in the time it takes to receive an acceptable quality mesh, and outperforms transmission via strict UDP in the quality of the final mesh.

The paper is organized in the following manner. In Section 2 we place our work in the context of what has been done in this area and discuss the different transport protocols used. Section 3 discusses our hybrid transmission protocol. Section 4 discusses the experimental setup used to conduct our tests, and provides details of the experiments we ran. Section 5 presents the results of those experiments with an analysis of them. Finally, in Section 6 we discuss our results and future work we intend to investigate.

2. OUR WORK IN CONTEXT

The PM scheme was devised by Hoppe in [17, 19], and we have implemented the basic version without the modifications of later work [18, 26]. This section introduces Hoppe's notation for the PM method that is relevant to our work; for a complete discussion, the reader is referred the works mentioned previously. The PM representation of a mesh M is stored as a coarse mesh M^0 and a sequence of n detail records called *vertex splits*. These vertex splits indicate how to incrementally refine M^0 so that after the n vertex splits have been processed, the original mesh M is recovered. In fact, the PM representation defines a sequence of meshes $M^0, M^1, \ldots, M^n$ which provide increasingly accurate approximations of M.

A vertex split is a basic transformation that adds a vertex to the mesh. The basic progressive mesh scheme is implemented, but for the purposes of this paper, a vertex split does not contain normal, texture, or material information. Each vertex split is a 30 byte quantity consisting of a face index, $flclw$, an index vs_index ($0 \leq vs\_index \leq 2$), an encoding vlr_rot, and two vertex position deltas, vad_l and vad_s. In our experiments, these vertex splits are packed into 1400 byte packets. Thus, each packet contains roughly 46 vertex splits.

Figure 1 illustrates a PM vertex split transformation. Each vertex split operation introduces a new vertex v_t and two new faces, as shown. The location of a vertex split is parameterized by v_s, v_l, and v_r. By default, the PM data structure does not include incidence information in the vertex to face direction. The vertex values are determined through the fields $flclw$, vs_index, and vlr_rot. To determine the vertex being split (v_s), the three vertices of the face $flclw$ are sorted by their index values and stored into an ordered list. They are indexed by vs_index. Vertex v_l is the next vertex clockwise on the face $flclw$. The vertex v_r is determined by vlr_rot, the number of clockwise rotations about v_s from v_l to v_r.

Other methods for progressive transmission of meshes have been developed. A more complicated scheme involving decomposing a mesh into a set of overlapping ellipsoids and point sampling the shape has been proposed by by Bischoff and Kobbelt [5]. We are currently evaluating their method in the context of our experiments, but we expect similar results since their method allows one to selectively refine areas of the mesh for which information was received, and not refine areas for which information was lost. Chen and Nishita [9] have constructed a streaming mesh format for progressive transmission with quality of service (QoS) control. This QoS control gives them advantages in anticipating network congestion, but their transmission technique is still built upon TCP and suffers the drawbacks of it. Using forward error correction for *compressed* progressive meshes [23] has been investigated by Al-Regib and Altunbasak [1]. Conceptually, this work is similar to our own. However, Al-Regib and Altunbasak consider the mesh as being decomposed *a priori* into discrete levels of detail. If the bit stream of one of these levels of detail is lost, there is an amount of distortion introduced into the rendered mesh corresponding to these levels of detail. Our method, in contrast, is more opportunistic in that it may lose more data, but renders what it can at a finer granularity, determined by the amount of data lost. An additional difference is that we have tested our method on a wide area network with simulated background traffic, whereas their results come entirely from simulations.

In the context of progressive transmission much of the work in the graphics community has focused on methods for better compression of geometric data rather than on robust transmission. Taubin et al. [32] achieve higher compression ratios than PM at the expense of a more complex refinement operation. Pajarola and Rossignac [23] refine the mesh in batches, which allows them to compress these batches effectively at the expense of some approximation quality advantages of the PM and Taubin methods. Khodakovsky et al. [20] achieve much better compression by remeshing the model into a semi-regular mesh. They then generate refinements through a wavelet transform and entropy encoding. Alliez and Desbrun [3] employ a valence driven decimation algorithm to achieve higher compression. Finally, Gandoin and Devillers [15] use a kd-tree algorithm that allows them to encode volumetric information as well as non-manifold meshes better than previous results. All of these techniques are complementary to our own, in that they could fit within the techniques proposed here to achieve higher transmission rates.

2.1 Transmission Control Protocol

In this section, we review the transmission control protocol (TCP) and its effect on network transmission over congested networks. Only material that is pertinent to this application is presented; a more thorough description can be found in books written by Stevens [31] or Comer [10]. In particular, this section looks at why using TCP, a reliable transport protocol, to transmit data over congested links results in significant delays and even unreliable service.

Using wide area networks (WANs), such as the Internet, to transmit data introduces significant delays (30ms-300ms) and loss of data during transmission. These delays affect the maximum rate at which a TCP connection can send information. The packet loss also limits the rate of transmission and can result in exponential delays in the transmission of data.

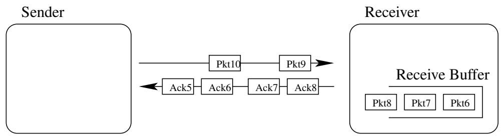

Figure 2: TCP Flow Control

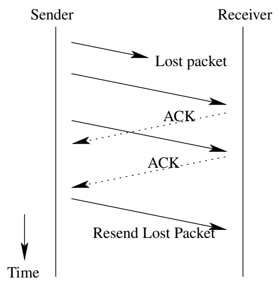

Figure 3: TCP Fast Retransmit

The limit on transmittable data over TCP occurs because of the interaction between long round-trip-times between sender and receiver and TCP's flow control algorithm. TCP uses flow control to guarantee that the sender does not overwhelm the receiver with data. This behavior is necessary so that a fast sender sending a packet every tenth of a second, for example, does not fill up the buffers on a slow receiver that only receives from those buffers every second. These buffers can be defined by the application. Without modifications to TCP, such as window-scaling, the maximum size of these buffers is 64 kB. This size limits how fast TCP sends during congestion. Congestion increases the round-trip time required to send a packet to the receiver and receive an acknowledgement. TCP limits the transmission rate so that a sending application does not send information when the receiver's buffer is full. For example, consider Figure 2. If the buffer size of the receiver is six packets, the sender cannot send any additional packets until it receives an acknowledgment from the receiver. This first decrease in performance limits the maximum TCP performance. In applications that transmit 3D geometry, this feature of TCP is not needed since the application should easily be able to consume information quickly enough that buffers will not fill up at the receiving host.

The second decrease in WAN performance is due to lost packets. TCP uses acknowledgments to determine when packet loss occurs. The sender will send the data and when the receiver receives a packet it will send an acknowledgment that indicates the smallest sequence number that the receiver has not received. TCP uses these acknowledgments to detect packet loss through either the absence of an acknowledgment or the fast retransmission algorithm. If the packet has not been acknowledged when the timeout occurs, the sender will resend that packet. The timeout must be large enough that a packet that is not lost would arrive at the server and be acknowledged before the timeout expires. TCP dynamically adjusts the timeout when it does not receive an acknowledgement. Typical network stacks exponentially increase length of the timeout. Because congested networks often cause several packet losses to occur consecutively, this can result in large delays, e.g., thirty seconds or more.

Fast retransmission recognizes that the sender knows a packet was lost because it receives duplicate acknowledgments for a previous packet. Thus, fast retransmission can avoid delays incurred waiting for timeouts. An example is shown in Figure 3 where the two packets sent after the lost packet include an acknowledgment number to the beginning of the lost packet. These duplicate acknowledgments occur because whenever a packet arrives TCP will acknowledge the packet and indicate the next byte it expects to receive, which will be the lost packet in this example. As a result, the sender can take whatever actions need to be performed when that packet has been lost. The TCP specification states that when three repeated acknowledgments have been received, the sender can resend the previous packet.

Lost packets not only require retransmission of the lost packet but also affect the transmission rate of the server. The TCP congestion control algorithm uses packet loss to determine when a network link is congested. When a packet is lost, TCP halves the number of packets that it sends across the network before expecting an acknowledgment. Therefore, packet loss will decrease the sending rate of the TCP sender and affect the transmission performance over congested networks. Combining the effect of reduced transmission rates and exponential timeouts results in significant delays that may even cause the failure of the network link between the sender and receiver even though the physical link still exists and unreliable transmission would succeed.

2.2 Real-time Data Transmission

Many papers have explored how to reliably send data through unreliable networks [27, 16, 13, 30]. Essentially, there are two approaches. In the first case, one adds additional information to packets such that the sender and receiver can determine if packets have been lost and the receiver can request a retransmission of any lost data. In the second case, one adds redundant information to packets such that the receiving process can reconstruct lost packets from data contained in other packets that this process has already received. The drawbacks of using TCP are that it increases the time of transmission of data and cannot scale for the use in multicast transmissions. An example of the second case is forward error correction codes. The primary disadvantage of these techniques is the addition of redundant data in the transmission.

Research into the transmission of multimedia streams [25, 28] has considered techniques that dynamically adjust the amount of information encoded in the multimedia stream to meet the bandwidth available between the sender and the receiver. In many cases, these schemes cannot afford to drop packets or use forward-error correction mechanisms. This approach differs from the transmission of 3D geometric information that we study in this paper. Alternative research in multimedia delivery systems has explored the use of transmission over partially ordered transport protocols [11] and increased use of buffers for supporting long-lived multimedia streams [2].

Additional research has investigated whether one can guarantee the quality of network transmissions. Guaranteed quality has an obvious and important impact on mesh transmission. Specific work includes integrated and differentiated services [33, 7, 6] within the Internet. These proposals provide mechanisms that could be used to control the number of packets lost in the network. As a result, the use of these mechanisms could eliminate concerns about lost packets. Unfortunately, deployment and use of integrated or differentiated services is not currently available.

3. HYBRID TRANSMISSION

In this section, we discuss our hybrid protocol for transmission of mesh data. The progressive mesh format creates an alternative representation of the 3D geometry. One significant advantage of this representation is that some packets containing mesh data are

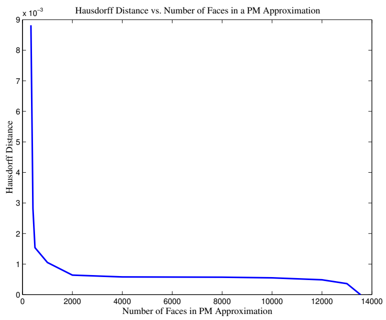

Figure 4: The Hausdorff distance of the Cessna model for different PM approximations by number of faces in the model.

more important than other packets. An example of this difference in packet priority appears in Figure 4. This figure compares the Hausdorff distance between the original 3D model and a model derived by performing a subset of the vertex splits on the base model. The Hausdorff distance is a standard way to measure the difference between two surfaces, and calculates the farthest distance from a point in one surface to its closet point in the other surface [21]. We calculate the surface deviation using the algorithm described in Aspert et al. [4]. Notice in the graph that the initial splits provide significant improvements in the Hausdorff metric, while later splits provide decreasing benefits.

The method implemented here has one addition to the basic PM strategy. In addition to the vertex splits, each packet contains a four byte header that stores the index number of the first face of the mesh that will be introduced by the first vertex split in this packet. The client renderer uses this number as a "poor man's" error correction, to give faces generated by vertex splits after lost packets their index number in the full resolution mesh. This process is done so that future splits whose face index, $flclw$, references these faces will be able to find them. This header is a monotonically increasing number, and can also be used to detect out-of-order packets. This scheme is simple, and allows us to reconstruct much more of the mesh than if this header were not included.

Our hybrid technique leverages the inherent differences in the effect that vertex splits have on the resulting visual accuracy of the reconstructed mesh. Thus, in this scheme, we begin by transmitting data using the TCP protocol. Part-way through the transmission, the hybrid sender closes the TCP connection and transmits the remaining data using UDP. This allows the sender to reliably transfer the base mesh and some of the initial splits and use a more aggressive technique to transfer less important splits.

TCP controls the rate that packets are sent to maintain flow control and minimize congestion. As a result, TCP does not provide the end-user control of the sending rate. When we consider using UDP for transmission of data, the end-user has significant influence over the send rate. Therefore, our application must carefully select the send rate. If the send rate exceeds the capacity of the channel, we will increase the number of packets lost. However, if we decrease the send rate, we will not lose any packets but the transfer of information will take much longer than a TCP connection might take. We experimentally determine UDP send rates to use in applications. In future work, we will investigate automatic mechanisms for setting the UDP send rate.

An issue of concern in implementing this idea is that the sender can begin sending UDP data to the receiver while the receiver is still reading data from the TCP connection. This behavior causes a problem since the initial data will fill up the operating system's buffer for incoming network data and start dropping UDP packets that arrive. Therefore, the hybrid protocol adds a delay before sending the UDP data until the receiver has received all TCP data.

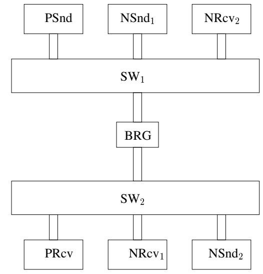

Figure 5: Testbed Setup

4. EXPERIMENTAL DESIGN

In this section, we present the design of an experimental testbed to explore the use of TCP and several hybrid protocols (TCP/UDP) to transmit 3D geometries of two different models across a congested network. We use a network testbed that allows us to emulate the network behavior of congested links. The testbed consists of a bridge machine and four hosts used to generate background noise traffic across the bridge.

4.1 Testbed Setup

Figure 5 shows the setup of our testbed. Our test environment consists of two user machines (PSnd, PRcv), four background load machines (NSnd_1, NRcv_1, NSnd_2, NRcv_2), one bridge machine (BRG) and two 100 Mbps switches (SW_1, SW_2).

	Processor	Memory	OS
PSnd, PRcv	800 Mhz AMD[1]	256 MB	Redhat 7.3
NSnd_1, NRcv_1	800 Mhz AMD	256 MB	Redhat 7.3
NSnd_2, NRcv_2	800 Mhz AMD	256 MB	Redhat 7.1
BRG	1 Ghz Pentium III	512 MB	FreeBSD 4.3

Table 1: Specifications for machines used in experiments.

The bridge runs the FreeBSD operating system and its kernel is compiled with options Bridge, Dummynet, HZ=1000 and NMBCLUSTERS=10,000. The Bridge and Dummynet options allow the use of this machine to emulate a WAN characteristics for packets sent between SW_1 and SW_2. The HZ option improves the resolution of the operating system's timer. The NMBCLUSTERS option is set prevent kernel from running out of mbuf clusters.

The parameters `rmem_max` and `wmem_max` in a Linux networking core are changed from 64kB to 8 MB. These parameters specify the maximum amount of memory a socket read/write buffer can use, respectively. Increasing them allows sockets to specify larger buffer sizes than the default maximum of 64kB. PSnd and PRcv enable TCP window scaling and set the read/write buffer size to 8 million bytes. This option reduces performance limits introduced by flow control as described in Section 2.1. Most user applications would not have the flexibility to reset the system settings and therefore would be limited to using less bandwidth. In our experiments, we have tuned the TCP system to obtain the best possible results for wide-area network transmission.

[1] AMD/Duron Processor

As shown in Figure 5, PSnd, $NSnd_1$ and $NRcv_2$ are connected with SW_1. PRcv, $NRcv_1$ and $NSnd_2$ are connected with SW_2. The bridge routes packets between SW_1 and SW_2.

4.2 Emulated WAN

The setup described above emulates a WAN. The WAN is emulated for several specific reasons. First, it provides an environment in which repeatable experiments can be conducted. If this were deployed over a true wide-area network, two different experiments could have significantly different amounts of packet loss and queuing delays. An alternative approach would be to use a network simulator or model to analytically evaluate the performance of different transmission methods. The analytic technique suffers from the disadvantage that many network models fail to adequately capture the underlying characteristics of real networks and the interactions of complex protocols such as TCP.

Dummynet [29] provides a standard implementation for emulating WANs. It allows users to create pipes of communication with specified bandwidth limits and propagation delays. These pipes are defined by matching header information such as source and destination IP addresses.

In all experiments, Dummynet limits the bandwidth and adds propagation delays to packets sent between switches SW_1 and SW_2. Specifically, we create two pipes, each with a bandwidth limit of 50Mbps and 25ms propagation delay. Most WAN communication will involve much longer propagation delays. As a result, we have erred on the low end, which will improve the performance of the TCP transmission in our experiments. The traffic from PSnd, $NSnd_1$ and $NRcv_2$ shares one pipe and the traffic from PRcv, $NRcv_1$ and $NSnd_2$ shares the other. Two pipes are necessary to emulate the out-bound and in-bound bandwidth between the sender and receiver. If the noise generated is not greater than the channel capacity, then packets will not be lost on the emulated network. In other words, a congested network requires more data to be sent than there is channel capacity available.

4.3 Background Load

The background load generator consists of two UDP components, a sender and a receiver. The background load generator simulates the network characteristics of a large number of external applications, communicating over a shared data link. Previous research has established that the time between packets arriving at a router matches a Pareto distribution [24]. This distribution is *heavy-tailed*, in that most of the inter-arrival times are small but there are periods of significant delay. The Pareto distribution creates self-similar behavior in the network, i.e., there are time scale independent bursts of packet activity.

Each of the two pairs of background load machines has a noise sender and a noise receiver. The noise sender sends UDP packets of size 1400 bytes to the the noise receiver, sleeping for a random amount of time before sending the next packet. The random time it waits follows a Pareto distribution, which favors shorter wait times. This effect causes the packets to have a higher probability of being sent in bursts than evenly spaced.

5. EXPERIMENTAL RESULTS

We tested our transmission methods by running a set of experiments with varying levels of background load using a variety of models. We report results for two models, the "happy Buddha" and a model of a Cessna. The Buddha model contains 1.08M faces in its full model and 1998 faces in its base mesh. The Cessna model has 13546 faces and 338 in its base mesh. Experiments involving the Buddha model consist of six runs of the experiment. Experiments involving the Cessna model consist of ten runs of the experiment. We present typical results in this section and discuss them, but graphs of the results of all experiments can be found in the Appendix. The results are discussed in terms of the average values across a set of runs. Some experiments timed out, i.e., they did not complete, so the average is the average of the experiments that completed in under 200 seconds. In the experiments, we measure the time taken to transmit the progressive mesh, the actual number of vertex splits received by the receiver, and the number of faces in the final mesh reconstructed by the receiver. The number of faces is important because some of the vertex splits that arrive may be unusable because of a previously dropped packet. We also compute the Hausdorff distance from the reconstructed meshes to the original full mesh. Finally, the suite of experiments was run using different send rates for the UDP portion of the model transmission.

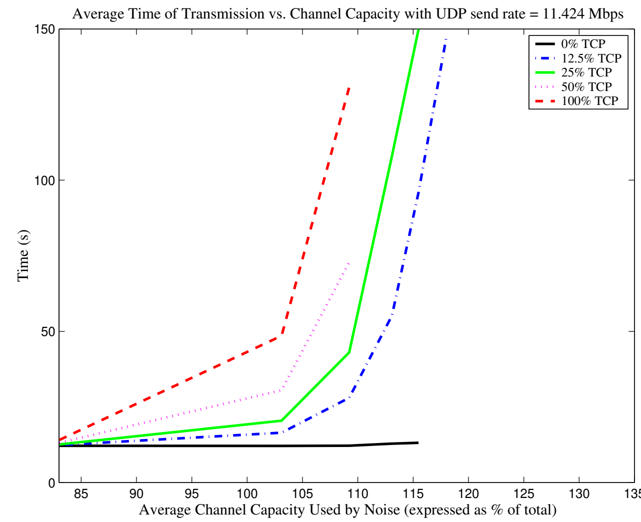

Figure 6: The average time of transmission of the Buddha model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 11.424 Mbs.

5.1 Results for Buddha Model

Figure 6 shows the average transmission time for the various transmission schemes, and Figure 7 shows the average number of faces received versus channel capacity. In these figures, the UDP send rate was 11.424 Mbps, and TCP was used to transmit the base mesh plus 0 to 50% of the rest of the model. Note that the lines have different length because at certain points, none of the experiments for that data point completed within the timeout period. The important result in Figure 6 is that the transmission time to send the model using pure TCP is the longest. Moreover, as the ratio of packets sent by TCP decreases, the transmission time improves for all given levels of noise. To explain this behavior, one must consider what happens when we increase the amount of noise. When the noise exceeds the channel capacity, the channel is full, and buffers in the network become saturated. This saturation results in packet loss. As discussed in in Section 2, packet loss causes TCP to incur timeouts or apply the fast retransmission scheme. Thus, the transmission time using TCP increases. As a final note, the 0% TCP plot in this figure is not quite constant because, as mentioned, the base mesh is transmitted reliably using TCP.

This transmission time improvement comes at a cost, however. Figure 7 shows the number of faces in the received model for each

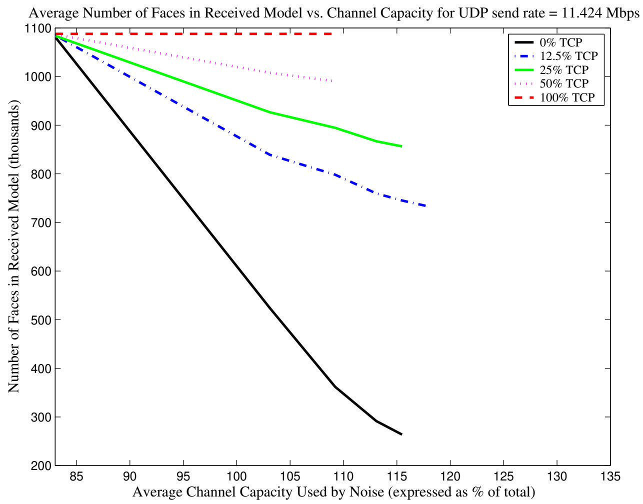

Figure 7: The number of faces received of the Buddha model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 11.424 Mbs.

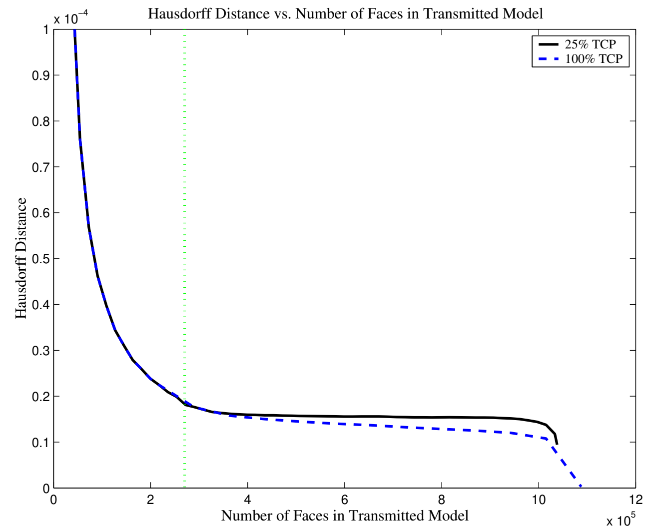

Figure 9: The Hausdorff distance versus number of faces in the Buddha model for the cases when the model is transmitted completely by TCP and 75% of the model is transmitted via UDP. The UDP send rate was 2.285 Mbs and the network load was 110% average channel capacity. The dotted line indicates the points where the graphs diverge, i.e., where 25% of the model has been transmitted via TCP.

of these transmission methods. Note that TCP, since it is a reliable transport protocol, always transmits all the faces. When a portion of the packets are sent using UDP, significant packet loss can occur, and this loss becomes worse as the noise increases. Therefore the selection of an appropriate hybrid protocol will depend on the tradeoff between the transmission time and the visual degradation that occurs when packets are lost. The visual degradation of this process is shown in the Buddha model in Figure 8. This figure shows the models that were received in one experiment using a UDP send rate of 11.424 Mbps and where the noise consumed 109% of the channel capacity. In particular, the visual quality of the model is excellent when 50% of it is transmitted via TCP. TCP took on average 131 seconds to transmit at this noise level, while the average transmission time for the 50% scheme was 73 seconds, a considerable savings. Hausdorff distances between the full model and these models are shown in Table 2.

In Figure 9, we see an example of the benefit of sending the model via the hybrid transmission scheme. The graph plots the Hausdorff distance between a PM representation with the indicated number of faces transmitted and the full model. The graph shows a model where only 25% of it is transmitted via TCP, and one transmitted completely via TCP. The dotted line indicates the point at which the graphs will begin to diverge. Both are monotonically decreasing. It takes on average 120 seconds to send the model fully via TCP, whereas the hybrid scheme's average transmission time is only 70 seconds. Thus, we get improvement in the model as measured by the Hausdorff distance at a savings of 50 seconds in transmission time. Note that sending 25% of the model via TCP takes on average 36 seconds, and sending 50% of the model via TCP takes on average 70 seconds. These results show that to have a more accurate model transmitted by TCP takes considerably longer, almost 50 seconds. Alternatively, the number of faces in the model must be reduced by half for pure TCP transmission to transmit in the same amount of time as the hybrid method, and the fidelity of the pure TCP model thus transmitted is lower.

A significant concern in any transmission scheme is that the bandwidth of the scheme is minimal. In Figure 10, we compare the number of bytes used to transmit the Buddha model by both TCP and UDP as we increase the amount of background noise. UDP is constant by design—it is equivalent to the TCP 0% line in the previous figures. Note that TCP uses more bandwidth as the noise increases, from 1.3 to 4.6%. This result is unsurprising since TCP must re-send lost packets, requires the use of acknowledgments, and has a larger header size than UDP packets. Note that the comparison in Figure 10 does *not* include the bandwidth of packets used to acknowledge receipt of data by the PM receiver.

5.2 Results for Cessna Model

Analogous to the previous section, Figure 11 shows the average transmission time of the Cessna model for the various transmission schemes, and Figure 12 shows the average number of faces received versus channel capacity. In these figures, the UDP send rate was 7.616 Mbps. The visual degradation of the models can be seen in Figure 13. These results are similar to those for the Buddha model,

% Sent	UDP Send Rate (Mbs)			
TCP	15.232	11.424	7.616	2.285
	Buddha Model			
0	6.5300	2.4100	4.2400	2.4200
12.5	0.1360	0.1510	0.1020	0.0899
25.0	0.0475	0.0470	0.0360	0.0237
50.0	0.0236	0.0195	0.0175	0.0113
	Cessna Model			
0	2.1095	2.2971	1.5034	2.6559
12.5	2.3495	1.1153	0.9845	0.5481
25.0	1.2377	2.3151	1.3259	0.4895
50.0	2.0762	0.7406	2.0183	1.5599

Table 2: Hausdorff distance between transmitted model (Buddha first, followed by Cessna) and full model for various transmission schemes (TCP/UDP ratios) in units of 10^{-3}.

Figure 8: The Buddha model after transmission over a lossy network: (a) the full model as sent by TCP; (b) model received when 50% transmitted by TCP; (c) model when 25% sent by TCP; (d) model when 12.5% sent by TCP; (e) model when fully sent via UDP.

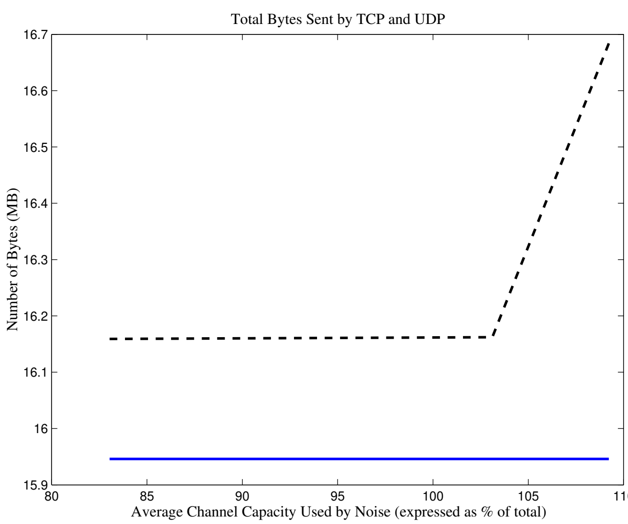

Figure 10: Total bytes sent by TCP (dashed line) and UDP (solid line) versus noise in the channel. Note that the UDP send rate is constant, but the number of bytes sent by TCP increases as it must resend data due to lost packets.

however there is considerably more variance in the results. This variance is largely due to the significantly smaller number of TCP packets transmitted, because the Cessna model is roughly two orders of magnitude smaller than the Buddha model. As a result, the base mesh for the Cessna model only consists of three or four packets. Therefore, if the initial SYN[2] request is lost, the results can be significantly skewed. This skewing occurs because the Linux TCP implementation significantly increases the packet timeout value if one of the SYN packets was lost. Therefore, when an additional packet is lost during transmission, a long timeout occurs (on the order of five to fifteen seconds) that significantly effects the results. If an initial packet is not lost, the timeouts will only be on the order of 400ms. Note that we do not count the connection setup time in our results, therefore our results do not include the SYN timeouts.

5.3 Discussion of Image Results

Both the Buddha and Cessna models have uncharacteristic flaws in the reconstruction. These flaws are apparent in Figures 8 (d) and (e) and in Figures 13 (c) and (d). Visually, the flaws appear to be missing triangles. However, these flaws are an effect of self-intersections of the mesh surface, i.e., certain triangles piercing the model. The inward facing normals of these triangles make the flaws noticeable. The surface self-intersections are the result of the PM encoding scheme and the lost transmission packets.

Since vs_index is the *relative* order of the index value of v_s among the three vertices on $flclw$, when packets are lost the reconstruction program occasionally fails and splits the wrong vertex. When a packet is lost, subsequent vertex splits may arrive at the receiver in different contexts from those in the original PM representation, i.e., the vertices adjacent to the split vertex v_s may be different. This information loss can cause geometric corruption, as shown in Figure 14. In Figure 14 (a), without packet loss, intermediate vertex splits change the three vertices of face f_{66} to 65, 96 and 114. A subsequent vertex split with $flclw$=66 and vs_index=1 splits vertex 96 because its index is the second small-

[2]TCP uses the SYN packets to initiate a request using a three-way handshake.

Figure 13: The Cessna model after transmission over a lossy network: (a) the full model as sent by TCP; (b) model received when 50% transmitted by TCP; (c) model when 25% sent by TCP; (d) model when 12.5% sent by TCP; (e) model when fully sent via UDP.

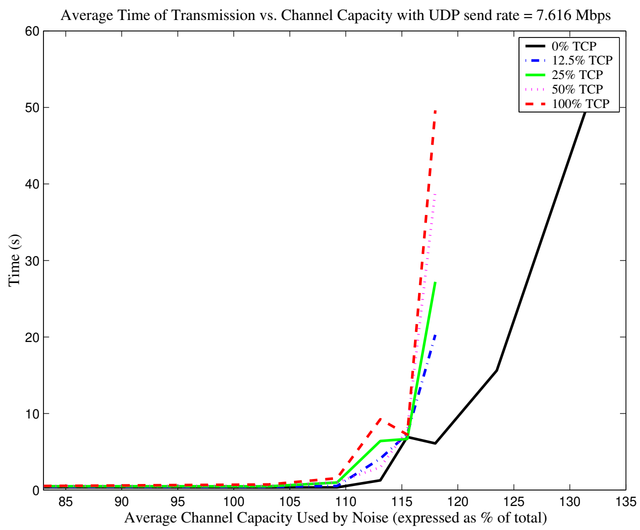

Figure 11: The average time of transmission of the Cessna model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 7.616 Mbs.

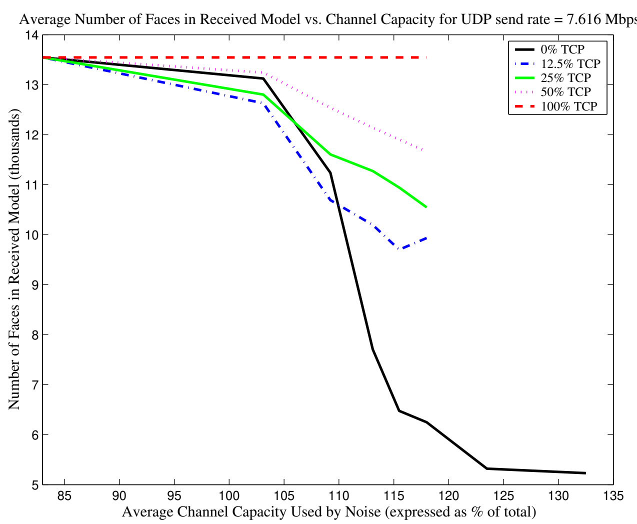

Figure 12: The number of faces received of the Cessna model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 7.616 Mbs.

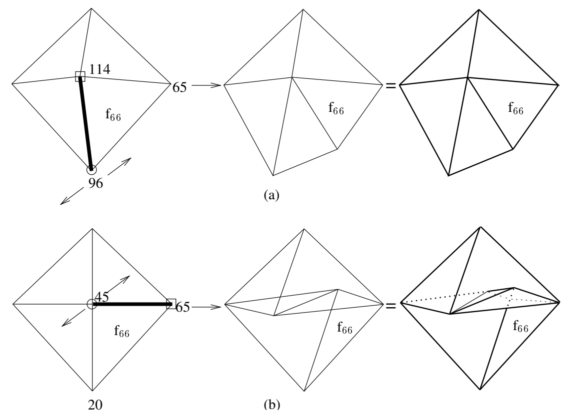

Figure 14: Illustration of self-intersection occurrence: (a) No packet loss, intermediate vertex splits change the three vertices of face f_{66} to 65, 96 and 114; (b) Packets containing the intermediate vertex splits are lost, the three vertices of face f_{66} remain unchanged as 65, 20, 45.

est. When the intermediate splits are lost, as shown in Figure 14 (b), the three vertices of face f_{66} remain unchanged as 65, 20, 45. Vertex 45 has the second smallest index among the vertices of f_{66} now. The same split with $flclw$=66 and vs_index=1 then splits vertex 45 and causes the faces connected with the added vertex to pierce through the top right face.

A pessimistic receiver would try to identify all out-of-context splits and throw them away. Implementing this pessimistic policy requires additional bandwidth to transmit the context information for each vertex split, however. Also, pessimistic policies will cause many splits to be discarded at moderate packet loss levels.

In our protocol design, we chose to minimize the additional information sent from sender to receiver. As a result, our reconstruction program discards splits whose $flclw$ fields reference faces that are not reconstructed because the vertex splits to reconstruct those faces are lost, and splits whose vlr_rot fields have values larger than the degree of v_s because previous splits to add faces connected with v_s are lost. All other received splits, which may include some out-of-context splits, are applied. This receiver behavior trades off potential corruption of the mesh against the potential improvement in the model when the vertex orderings are valid.

We are currently investigating ways to apply out-of-context splits without corrupting the mesh.

6. CONCLUSION

In this paper, we have demonstrated that using a hybrid approach to send 3D geometry over congested networks can improve the transmission performance. Additionally, the hybrid scheme can

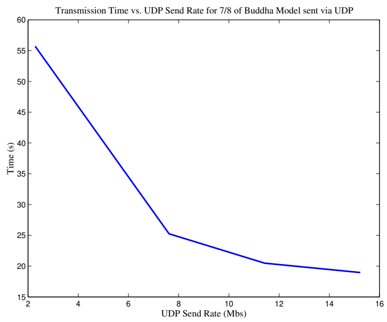

Figure 15: Transmission time of the the Buddha model with 87.5% of the model sent via UDP versus the UDP send rate.

result in a higher quality result than transmitting via TCP for an equivalent amount of time, as shown in Figure 9. This improvement in performance comes at the cost of lost packets. The progressive mesh representation allows us to minimize the visual impact when packets are lost, although some surface corruption is often visible. We are currently investigating methods of representing and reconstructing the progressive mesh that eliminate this corruption with less overhead than employing a forward error correction (FEC) code arbitrarily on the data.

There are many avenues for future research. One direction is to explore the refinement scheme of Bischoff and Kobbelt [5] and investigate its incorporation into our transmission protocol. Another direction is to determine how much model degradation users are willing to tolerate, and use this knowledge to select an appropriate UDP send rate for our hybrid approach. One significant result is that when transmitting small models such as the Cessna mesh, the poor performance of the TCP connection has a disproportionate effect on the total transmission time of the model. In future work, we plan on considering a FEC code that could be used to encode the base mesh. The FEC code would allow reconstruction of the base mesh even when some packets have been lost, and thus we may potentially abandon TCP as a protocol altogether.

Additionally, determination of an optimal UDP send rate is an interesting open problem. That an optimal UDP send rate exists is evidenced by Figure 15, where the transmission time of the Buddha model is shown for various UDP send rates, when 12.5% of the model is transmitted via TCP. We would like to automatically determine the knee of this curve. In particular, we would like to evolve the method into an adaptive scheme, where the sending mechanism automatically detects the network condition and optimizes the send rate for both the state of the network and the model in a TCP-friendly manner.

APPENDIX

A. COMPLETE DATA OF TRANSMISSION TIMES AND FACES

Figures 16 through 27 detail the average results for all the experiments run.

B. REFERENCES

[1] AL-REGIB, G., AND ALTUNBASAK, Y. An unequal error protection method for packet loss resilient 3-d mesh transmission. In *Proceedings of INFOCOM 2002* (2002).

[2] ALLEN, A. D. Optimal delivery of multi-media content over networks. In *Ninth ACM Multimedia Conference* (Ottawa, Ontario, Canada, 30 Sept.–5 Oct. 2001), pp. 79–88.

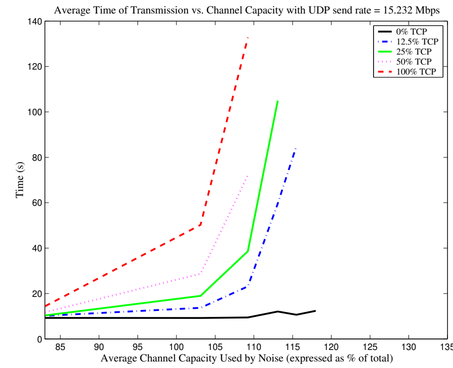

Figure 16: The average time of transmission of the Buddha model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 15.232 Mbs.

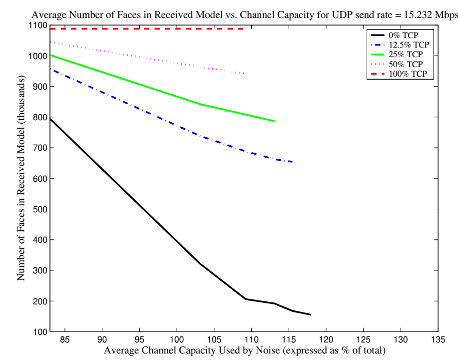

Figure 17: The number of faces received of the Buddha model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 15.232 Mbs.

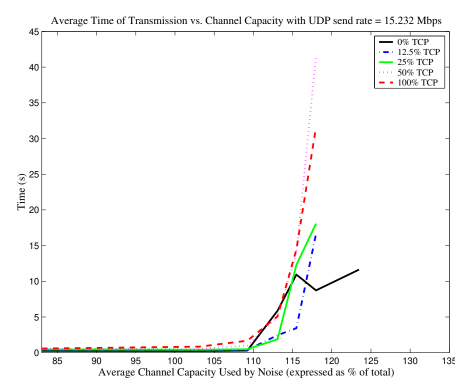

Figure 18: The average time of transmission of the Cessna model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 15.232 Mbs.

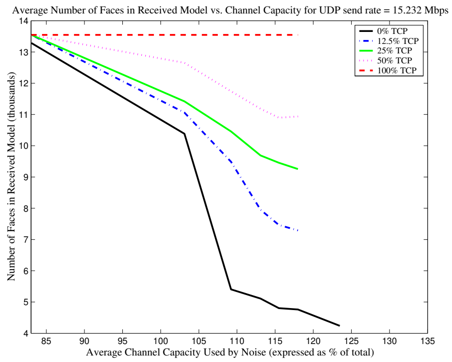

Figure 19: The number of faces received of the Cessna model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 15.232 Mbs.

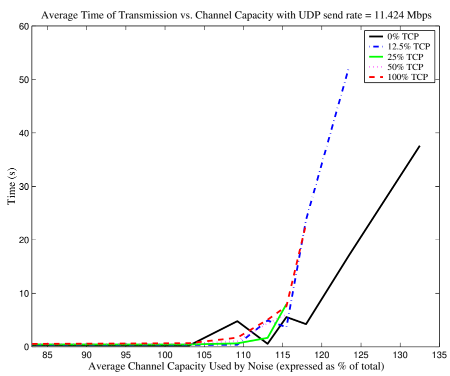

Figure 20: The average time of transmission of the Cessna model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 11.424 Mbs.

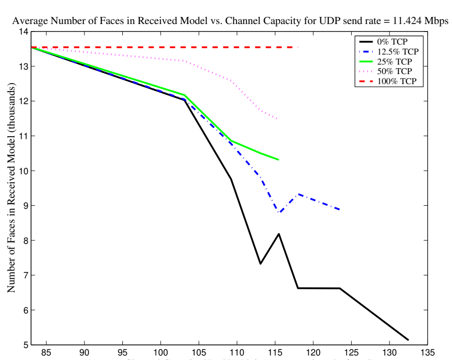

Figure 21: The number of faces received of the Cessna model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 11.424 Mbs.

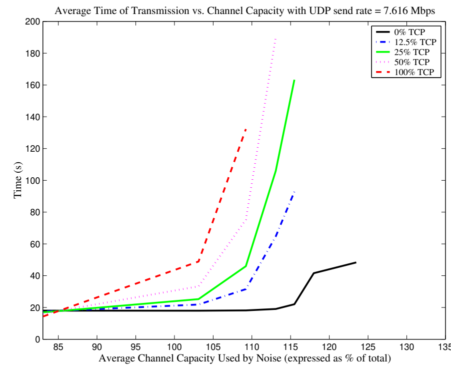

Figure 22: The average time of transmission of the Buddha model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 7.616 Mbs.

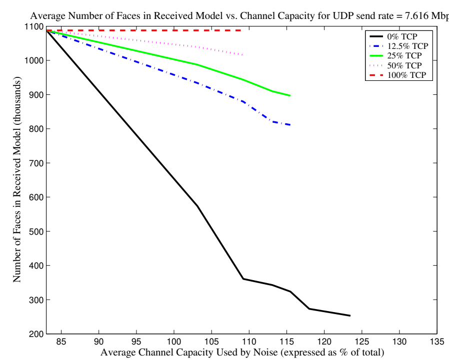

Figure 23: The number of faces received of the Buddha model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 7.616 Mbs.

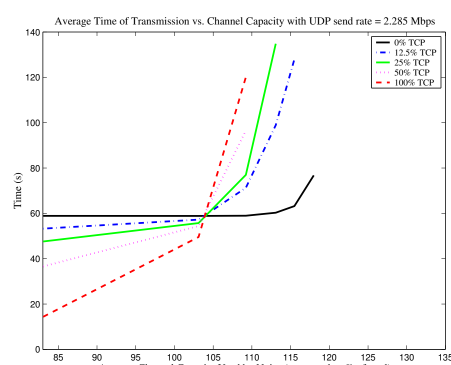

Figure 24: The average time of transmission of the Buddha model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 2.285 Mbs.

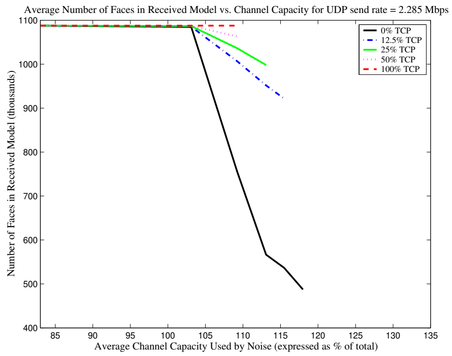

Figure 25: The number of faces received of the Buddha model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 2.285 Mbs.

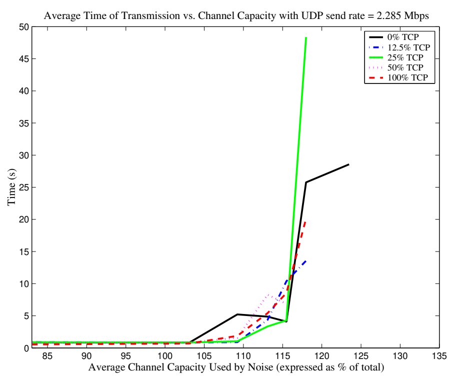

Figure 26: The average time of transmission of the Cessna model versus the channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 2.285 Mbs.

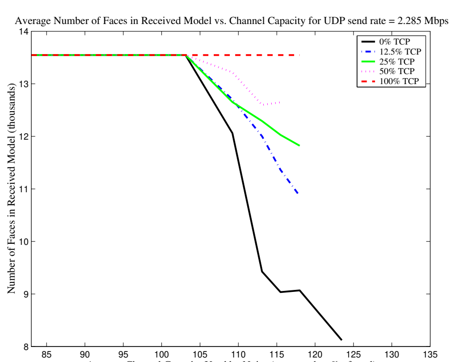

Figure 27: The number of faces received of the Cessna model versus channel capacity for the different transmission schemes (TCP/UDP ratios). The UDP send rate was 2.285 Mbs.

[3] ALLIEZ, P., AND DESBRUN, M. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of ACM SIGGRAPH 2001* (July 2001), Computer Graphics Proceedings, Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 195–202.

[4] ASPERT, N., SANTA-CRUZ, D., AND EBRAHIMI, T. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference in Multimedia and Expo (ICME)* (2002), pp. 705–708. http://mesh.epfl.ch.

[5] BISCHOFF, S., AND KOBBELT, L. Streaming 3d geometry over lossy communication channels. In *Proceedings of the IEEE International Conference on Multimedia and Expo* (2002).

[6] BLAKE, S., BLACK, D. L., CARLSON, M. A., DAVIES, E., WANG, Z., AND WEISS, W. An architecture for differentiated services. Request for Comments, RFC 2475, 1998.

[7] BRADEN, R., CLARK, D., AND SHENKER, S. Integrated services in the internet architecture: an overview. Request for Comments, RFC 1633, 1994.

[8] CASNER, S., AND DEERING, S. First IETF Internet audiocast. *ACM Computer Communication Review 22*, 3 (July 1992), 92–97.

[9] CHEN, B.-Y., AND NISHITA, T. Multiresolution streaming mesh with shape preserving and qos-like controlling. In *Proceeding of the 7th International Conference on 3D Web Technology* (2002), pp. 35–42.

[10] COMER, D. E. *Internetworking with TCP/IP, Principles, Protocols, and Architectures*, fourth ed. Prentice Hall, 2000.

[11] CONRAD, P. T., CARO, A., AND AMER, P. ReMDoR: Remote multimedia document retrieval over partial order transport. In *Ninth ACM Multimedia Conference* (Ottawa, Ontario, Canada, 30 Sept.–5 Oct. 2001), pp. 169–180.

[12] http://everquest.station.sony.com.

[13] FLOYD, S., JACOBSON, V., LIU, C.-G., MCCANNE, S., AND ZHANG, L. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking 5*, 6 (1997).

[14] FRÉCON, E., GREENHALGH, C., AND STENIUS, M. The DIVEBONE — an application-level network architecture for Internet-based CVEs. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)* (London, UK, 20–22 Dec. 1999), pp. 58–65.

[15] GANDOIN, P.-M., AND DEVILLERS, O. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics 21*, 3 (July 2002), 372–379. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[16] GARLICK, L. L., ROM, R., AND POSTEL, J. B. Reliable host-to-host protocols:problems and techniques. In *Proceedings of the Fifth Data Communications Symposium. Applications, Technologies, Architectures, and protocols for Computer Communication* (Snowbird, Utah, USA, 1977).

[17] HOPPE, H. Progressive meshes. In *Proceedings of SIGGRAPH 96* (New Orleans, Louisiana, August 1996), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 99–108. ISBN 0-201-94800-1.

[18] HOPPE, H. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH 97* (Los Angeles, California, August 1997), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison

Wesley, p. I198. ISBN 0-89791-896-7.

[19] HOPPE, H. Efficient implementation of progressive meshes. *Computers & Graphics 22*, 1 (February 1998), 27–36. ISSN 0097-8493.

[20] KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive geometry compression. In *Proceedings of ACM SIGGRAPH 2000* (July 2000), Computer Graphics Proceedings, Annual Conference Series, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 271–278. ISBN 1-58113-208-5.

[21] KLEIN, R., LIEBICH, G., AND STRASSER, W. Mesh reduction with error control. In *IEEE Visualization '96* (1996), R. Yagel and G. M. Nielson., Eds., pp. 311–318.

[22] MACEDONIA, M. R., ZYDA, M. J., PRATT, D. R., BRUTZMAN, D. P., AND BARHAM, P. T. Exploiting reality with multicast groups. *IEEE Computer Graphics and Applications 15*, 5 (Sept. 1995), 38–45.

[23] PAJAROLA, R., AND ROSSIGNAC, J. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (January - March 2000), 79–93. ISSN 1077-2626.

[24] PARK, K., KIM, G., AND CROVELLA, M. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of the Fourth International Conference on Network Protocols (ICNP '96)* (October 1996).

[25] PEJHAN, S., HAO CHIANG, T., AND ZHANG, Y.-Q. Dynamic frame rate control for video streams. In *Seventh ACM Multimedia Conference* (Orlando, FL, USA, 30 Oct.–5 Nov. 1999).

[26] POPOVIĆ, J., AND HOPPE, H. Progressive simplicial complexes. In *Proceedings of SIGGRAPH 97* (Los Angeles, California, August 1997), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 217–224. ISBN 0-89791-896-7.

[27] POSTEL, J. Transmission control protocol. RFC 793, `http://www.rfc-editor.org/rfc/rfc793.txt`, Sept. 1981.

[28] REJAIE, R., HANDLEY, M., AND ESTRIN, D. Quality adaptation for congestion controlled video playback over the internet. In *ACM SIGCOMM Conference. Applications, Technologies, Architectures and Protocols for Computer Communications.* (Cambridge, MA, USA, 30 Aug.–3 Sept. 1999), vol. 29(4) of *Computer Communication Review*.

[29] RIZZO, L. Dummynet:a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review 27*, 1 (Jan. 1997).

[30] SINHA, P., NANDAGOPAL, T., VENKITARAMAN, N., SIVAKUMAR, R., AND BHARGHAVAN, V. WTCP: A reliable transport protocol for wireless wide-area networks. *Wireless Networks 8*, 2/3 (Mar.–May 2002).

[31] STEVENS, W. R. *UNIX Network Programming, Networking APIs: Sockets and XTI*, second edition ed. Prentice Hall PTR, 1998.

[32] TAUBIN, G., GUEZIEC, A., HORN, W., AND LAZARUS, F. Progressive forest split compression. In *Proceedings of SIGGRAPH 98* (Orlando, Florida, July 1998), Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH / Addison Wesley, pp. 123–132. ISBN 0-89791-999-8.

[33] ZHANG, L., DEERING, S., ESTRIN, D., SHENKER, S., AND ZAPPALA, D. RSVP: A new resource ReSerVation Protocol. *IEEE Network 7*, 5 (Sept. 1993), 8–18.