A Network Architecture Supporting Consistent Rich Behaviour in Collaborative Interactive Applications

James Marsh, Mashhuda Glencross, Steve Pettifer, and Roger Hubbold

***Abstract*— Network architectures for collaborative virtual reality have traditionally been dominated by client–server and peer-to-peer approaches, with peer-to-peer strategies typically being favoured where minimising latency is a priority, and client–server where consistency is key.**

With increasingly sophisticated behaviour models, and the demand for better support for haptics, we argue that neither approach provides sufficient support for these scenarios and thus a hybrid architecture is required. We discuss the relative performance of different distribution strategies in the face of real network conditions, and illustrate the problems they face.

Finally we present an architecture that successfully meets many of these challenges, and demonstrate its use in a distributed virtual prototyping application which supports simultaneous collaboration for assembly, maintenance and training applications utilising haptics.

***Index Terms*— Virtual reality; Network Architecture and Design; Haptic I/O; Computer-supported collaborative work; Computer-supported cooperative work; Simulation, Modeling, and Visualization; Client/server; Distributed applications; Computer-aided design.**

I. Introduction

As the field of virtual reality matures, our expectations become increasingly ambitious. Having developed from simple demonstrators and networked games supporting basic behaviour to complex engineering applications in the areas of training, design, planning and product testing [1], [2], the support for complex behaviour is ever more important. With increasing processor speeds and sophisticated general purpose computation possible on Graphics Processing Units [3]–[5], the potential for computing such complex behaviour in real-time (including physical simulation) has greatly improved.

More realistic behaviour, however, brings with it the need for more intuitive modes of interaction, both in terms of input devices and the underlying algorithms. For example, with a number of researchers having demonstrated experimentally that providing kinaesthetic feedback through the use of haptic interaction devices has a statistically significant impact upon task-performance [6]–[11], advanced models for haptic interaction are being investigated to provide additional sensory feedback [12]–[14]. A user interacting with an object via a haptic device may quite reasonably expect to feel forces such as friction, or experience the textures of objects in a way that a mouse-user would not even consider. The extra computational and architectural cost of providing such an experience should not be underestimated. In spite of these extra complexities the scope for combining these technologies to enable remote collaboration is huge, with particularly compelling applications in computer aided design, such as for virtual prototyping and collaborative design reviews [15]–[17]. Whereas the fundamental techniques have continued to develop apace, in recent years network technology has been unable to match their rapid progress. While available bandwidth has been steadily increasing (with broadband DSL connections gradually replacing ISDN), high latency (the delay between the transmission of a message and its reception), and high jitter (the variation in latency) are still surprisingly common, and it is these that currently present the greatest challenge to sophisticated collaborative virtual environments.

There have been many recent proposals for architectures and frameworks to support haptically-enabled collaborative applications [18]–[23], however we argue in this paper that none of them have been flexible enough to support the kind of rich behaviour that we envisage here.

In this paper we present the results of our experience of developing such an architecture, beginning by considering the demands imposed on the system, the network conditions we experienced, and the considerations required to contend with them. Specifically, we illustrate our discussion with reference to a distributed interactive virtual prototyping system (DIVIPRO), incorporating haptics, geometric constraints and multi-body simulation, and which has been demonstrated to work successfully over a wide variety of network conditions.

II. Interaction in Behaviourally Rich Environments

The interaction modes adopted by users to achieve shared objectives in distributed virtual environments vary: at one extreme each participant may perform essentially independent tasks in pursuit of a common goal; at the other extreme multiple users can simultaneously interact with the same model. In the latter case this could represent either a single physical object, or form part of a complex multi-body simulation. One classification [18] of these types of interaction modes defines *collaborative* tasks as those in which participants collaborate by taking turns to achieve a particular shared objective, for example while performing an assembly sequence for virtual prototyping purposes; and *co-operative* tasks as those in which participants co-operate by simultaneously manipulating the same entities, such as in a carrying task.

The distinction between these types of activities is particularly useful when judging the requirements imposed upon distribution architectures. Similarly, the manner in which the environment is presented to the users imposes further demands upon the system. For example, as three-dimensional input devices and multi-sensory rendering techniques improve, users are more likely to notice the lack of visual and behavioural fidelity in shared virtual environments. Simple techniques such as texture and bump mapping used to increase visual fidelity require a corresponding haptic representation. If a haptic model feels unlike the user's expectation of the graphically displayed object then participants will be unconvinced of the realism of the environment. Worse still, if haptic feedback is over-simplified then there is a danger it could reinforce erroneous behaviour patterns that subsequently need to be 'un-learnt' when applied in the real world.

III. Factors Affecting Task-Performance in Virtual Environments

Many studies have examined the effectiveness of collaborative virtual environments and groupware applications. These have considered a number of factors that affect the extent to which such environments can be successful, including network latency, collaboration strategies, rendering fidelity, and interaction mechanisms.

Users have been shown to be able to adequately perform tasks in the face of fixed network latencies up to 200ms [24]. However this is highly task dependent, and often attributed to the users adopting a 'move and wait' strategy. Jitter potentially has a greater impact on the subjects' ability to co-ordinate their actions, particularly with regard to predicting the actions of their collaborators [25].

Changes in interaction strategy to compensate for the effects of latency and jitter could again reinforce erroneous behaviour. For example, if during a training application a network glitch causes a sudden and surprising artifact, the trainee may believe that they caused it by making an error, even though they were performing the correct operation.

For the visual display of 3D environments, one of the most important considerations is maintaining suitably high frame rates. Typically this is considered to be at least 20–30Hz [1]. Below this level motion appears discontinuous, and object interactions (particularly collisions) may fail to be represented correctly due to the high inter-frame latency.

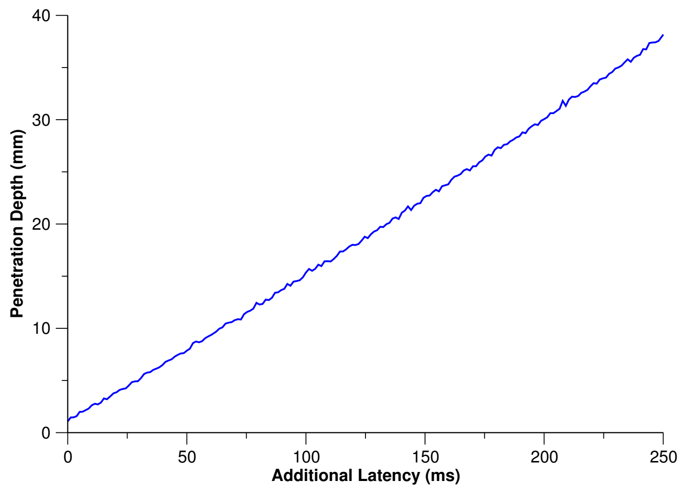


Fig. 1
The Effect of Communication Latency on Penetration Depth When Dropping a 50g Mass 10cm.

The addition of interaction through force feedback has been shown to improve skills transfer in single user training applications over the use of VR alone [8], and employing haptic feedback to support collaboration can considerably enhance both task-performance and co-presence [6]. However haptic rendering imposes even greater demands on the system than visual displays.

An important factor contributing to the correct perception of a collision with a solid, haptically-rendered surface is the amount by which the virtual surface can be penetrated. This is highly dependent on the latency inherent in the feedback system controlling the haptic device. Latency arises from two sources: the update rate of the device's feedback loop, and communication delays. An update rate of at least 1KHz [26] is considered to be necessary for solid contacts; below this rate objects begin to feel 'spongy', and if the rate drops too low, instabilities arise.

To quantify the typical effect of communication latency on haptic response we performed a simple experiment using one of our FCS HapticMASTERs [27]. These are 6 degrees-of-freedom (6 DOF) input, 3 DOF output, haptic interfaces which operate on an admittance control principle. The device provides a large workspace determined by the sweep of its robotic arm, with a maximum vertical extent of 0.4m.

The arm was programmed to repeatedly move to the same position from which a 50g mass caused it to drop under the influence of gravity 10cm onto a horizontal virtual plane. A constant update rate of 1KHz was maintained, however the force calculation was based on a delayed reading of the end-effector position and velocity. The maximum penetration depth for a range of latencies was then plotted (see Figure 1). Communication latency inherent in driving the device itself was ignored and assumed to be constant.

While penetration depth will depend on a number of factors, including surface stiffness and damping properties, force applied, and impact velocity, in informal experiments we found that users generally began to report collisions with surfaces feeling 'unusual' when the additional latency reached the 25–30ms range.

IV. Architectures Supporting Collaboration

The primary aim of any architecture designed to enable collaboration is to provide a consistent and coherent view of a shared environment or application to each of the connected users. Ultimately the speed of light imposes a finite upper bound on how fast information can be communicated across a network, and in practice real network performance is well below this level. The impact of this is that regardless of architecture, two remote users cannot share an instantaneous, fully-synchronised view of an environment.

Table I and Figure 2 show the result of measuring latency and jitter on a variety of different network routes between The University of Manchester and a number of our collaborators. The results show that the routes differ in performance by one or two orders of magnitude. Consequently the architectural choices which are appropriate for a local-area network (LAN) are not necessarily going to apply in the case of typical wide-area network (WAN), and likewise applications designed specifically to work over WANs are unlikely to take full advantage of the faster local environment.

Faced with this challenge, the majority of collaborative virtual environments adopt one of the two most common network distribution architectures, client–server or peer-to-peer [28], with both of these architectures having their own specific benefits and shortcomings.

A. *Client–Server — Increasing Consistency*

One of the most straightforward architectures to support complex behaviours, the client–server approach illustrated in Figure 4(a), uses a centralised server that runs a simulation of the virtual environment and communicates relevant state changes (including positional updates) to all of the attached clients. These contain a representation of the application, but do not perform any simulation activities locally; instead, user interactions are sent directly to the server for processing. The clients only update their local representations when they receive a message instructing them to do so.

This has the advantage that it makes simulating rich behaviour considerably more straightforward than alternative architectures. Instead of distributing the simulation itself across the remote clients, only the outcome needs to be communicated. The architecture is robust against the effects of jitter and latency since if update events destined for a particular client suddenly experience an increased delay, the effect will be no more serious than interference disrupting a television programme. The client may temporarily display an inconsistent view of the world, but as soon as the updates arrive this is corrected. Similarly, if updates destined for the server are delayed then events may be applied in a different order to originally generated. Unlike architectures employing multiple simulations, in this case all users will experience the same outcome. In many cases, given that the users have no other frame of reference, it would be impossible for them to determine the original chronology. While this may result in occasional perceptual problems (such as 'dead men shooting'), the server will remain in a semantically consistent state.

In addition to the synchronisation benefits, with a single point of contact, application start-up is simplified, and it is straightforward to have the environment persist even if all the clients disconnect. These properties are particularly advantageous when the time taken to complete a task is high.

The biggest disadvantage of the client–server approach is that the local view of the environment is only updated after a round-trip to the server, resulting in additional input latency for the participants. However it has been argued that this architecture is the most appropriate to support co-operative tasks in collaborative virtual environments due to the need for consistent state [18].

Recall the requirement for visual update rates of at least 20–30Hz. If user input cannot be processed and displayed at this rate then input could be seen to lag and the users' perceptions of causal relationships may begin to break down. What effect this will have is likely to be task dependent, and experiments disagree with regard to the threshold at which task-performance is affected.

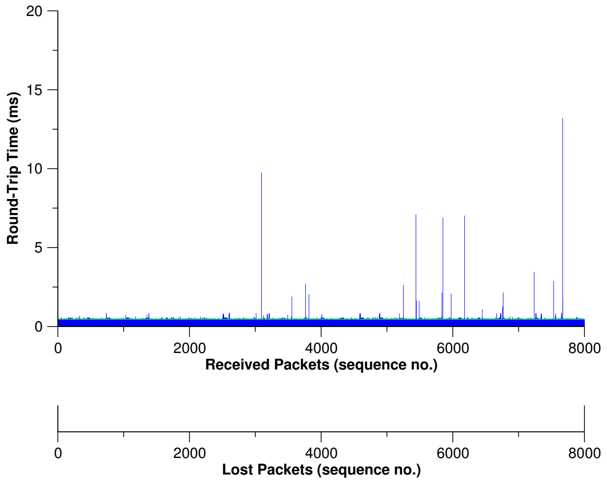
An analysis of the round-trip latencies of our office LAN, and the links between The Universities of Manchester and Bristol, and between The University of Manchester and FCS-Control Systems in Amsterdam (see Table I) shows that these network conditions would provide sufficiently high update rates that a pure client–server is likely to be adequate for a purely visual display (assuming whatever simulation processing was required could be calculated fast enough on the server). Where the graphs show occasional peaks of latency higher than 50ms occurring, depending on the task it may be possible to visually perceive brief glitches in continuous movement, though the global semantic state of the simulation would be unaffected. Haptic-rendering on the other hand demands much higher update rates than visual displays, and hence only a local LAN is able to provide low enough latency to make centralising haptic rendering feasible.

B. *Peer-to-Peer — Reducing Latency*

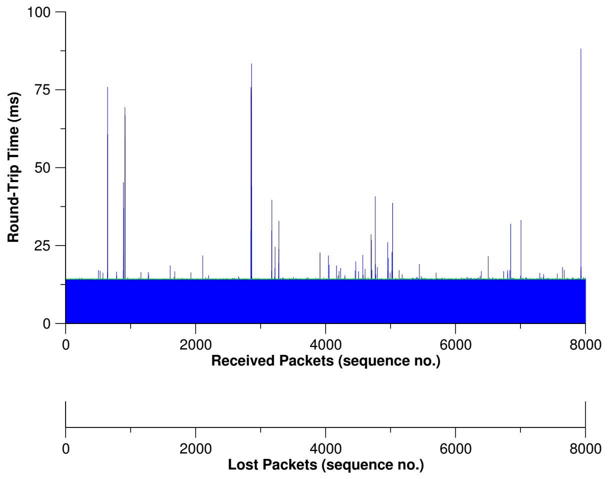
With a peer-to-peer architecture, peers directly apply a user's input to their own simulation of the environment while simultaneously communicating these events to other peers as shown in Figure 4(b). This is attractive because it avoids the additional input latency present with the client–server approach and hence has proved a popular architecture for a number of collaborative systems [29]–[32]. For applications with low synchronisation requirements (such as war games, viewing scientific datasets or static models, simple carrying tasks, and primarily social environments) such an architecture can be reasonably successful, especially over low-latency, low-jitter connections.

When more complex rich behaviour is involved, however, synchronisation issues begin to dominate. If simulation states computed by each peer diverge catastrophically (due to updates either being discarded, arriving in different orders, or at different times at each peer), in the absence of a server to arbitrate, correcting the causal inconsistencies that arise poses a significant challenge.

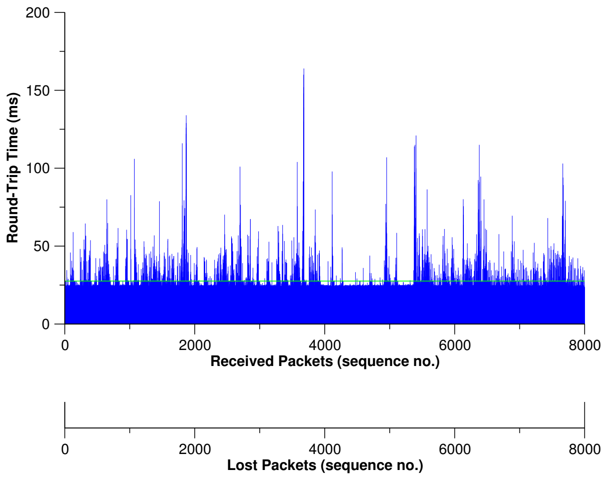
Most solutions to the problem of ensuring a consistent ordering of discrete events across a network of machines have evolved from the concept of 'logical clocks' [33]. While ensuring that events are executed in the same order at all peers, this particular algorithm is not tolerant of peers failing, and delayed messages cause all further processing to be delayed until the missing update is received. These problems have been addressed within subsequent work on virtual time and time-warping [34] which allows the simulation to be rolled back to a consistent state if delayed updates cause nodes to become in conflict. While these concepts have been suggested for use in virtual environments [35], [36], it appears the approach has been used in few real applications. Its main problem is that in addition to needing a history of simulation state to be stored, the ability to rapidly move forwards through time in order to 'catch up' to the current time is also required. This either implies significant spare CPU capacity or is likely to introduce further latency.



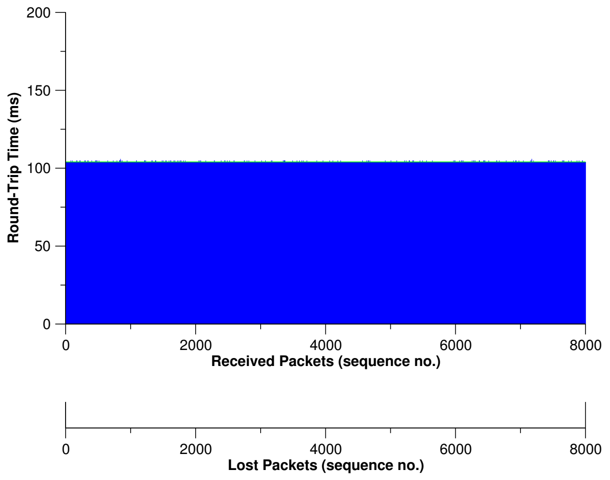
(a) Local LAN



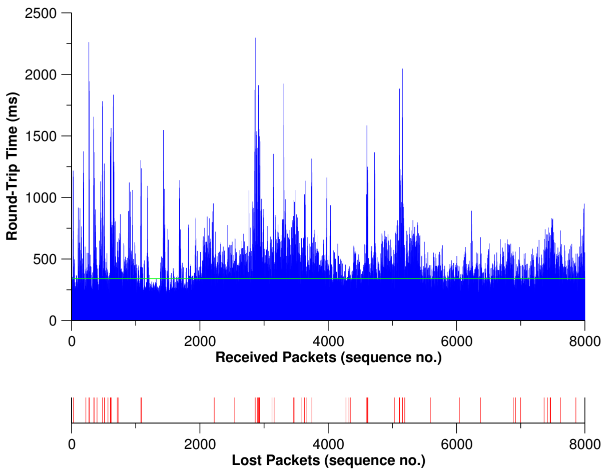
(b) The University of Bristol, United Kingdom



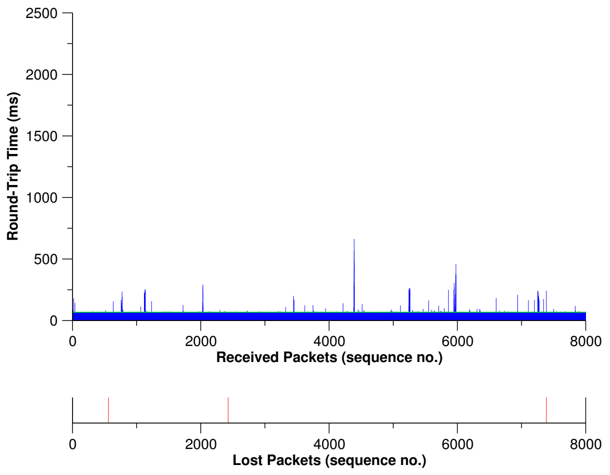
(c) FCS, Amsterdam, The Netherlands



(d) The University of North Carolina, Chapel Hill, USA



(e) Labein, Bilbao, Spain (afternoon)



(f) Labein, Bilbao, Spain (night)

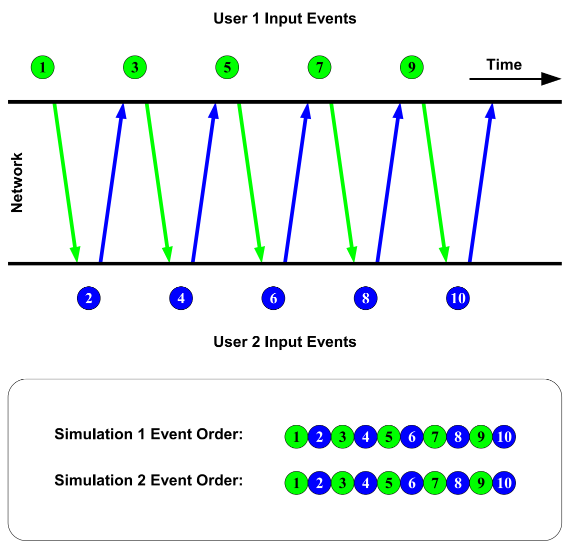
Fig. 2

MEASURED NETWORK PERFORMANCE TO VARIOUS DESTINATIONS FROM THE UNIVERSITY OF MANCHESTER, UK.

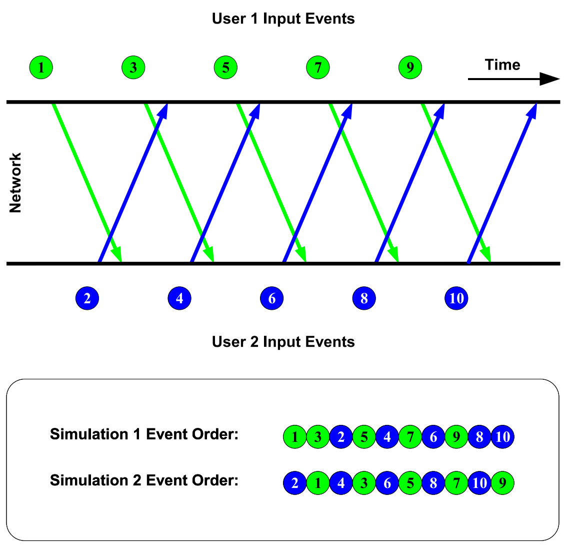
TABLE I

Results of Sending 8000 Packets to Different Destinations from The University of Manchester, United Kingdom.

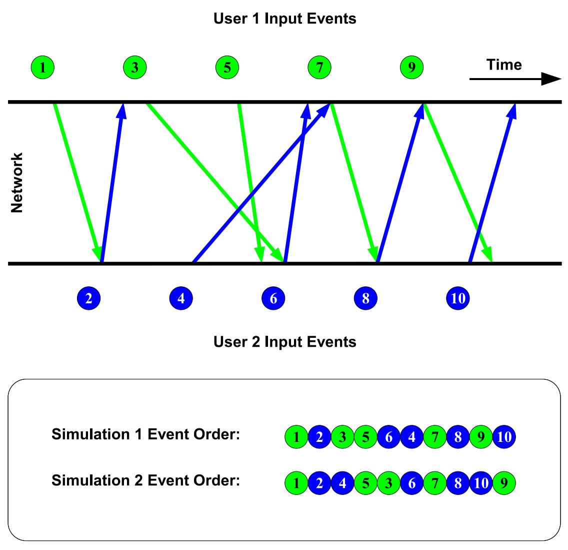
Destination	Distance (km)	Lost Packets	—— Round-Trip Time (ms) —— Min.	Max.	Mean	Mean Deviation	Mean Round-Trip Update Rate (Hz)	Typical Hop Count
Office LAN (Switched Ethernet)	0.01	0	0.36	13.2	0.46	0.047	2170	1
The University of Bristol, UK	230.00	0	13.9	88.2	14.3	0.34	69.9	14
FCS, Amsterdam, The Netherlands	496.00	0	20.9	164	27.6	4.54	36.2	18
Labein, Bilbao, Spain (night)	1140.00	3	62.3	663	68.3	4.59	14.6	17
University of North Carolina, USA	6062.00	0	104	106	104	0.038	9.62	18
Labein, Bilbao, Spain (afternoon)	1140.00	68	87	2298	341	136	2.93	17


User 1 Input Events
1 3 5 7 9
Time
Network
2 4 6 8 10
User 2 Input Events
Simulation 1 Event Order: 1 2 3 4 5 6 7 8 9 10
Simulation 2 Event Order: 1 2 3 4 5 6 7 8 9 10

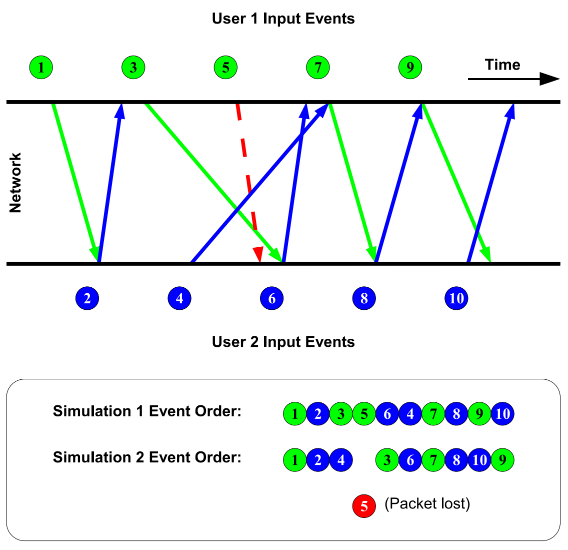
(a) Constant Low Latency


User 1 Input Events
1 3 5 7 9
Time
Network
2 4 6 8 10
User 2 Input Events
Simulation 1 Event Order: 1 3 2 5 4 7 6 9 8 10
Simulation 2 Event Order: 2 1 4 3 6 5 8 7 10 9

(b) Constant High Latency


User 1 Input Events
1 3 5 7 9
Time
Network
2 4 6 8 10
User 2 Input Events
Simulation 1 Event Order: 1 2 3 5 6 4 7 8 9 10
Simulation 2 Event Order: 1 2 4 5 3 6 7 8 10 9

(c) Latency and Jitter


User 1 Input Events
1 3 5 7 9
Time
Network
2 4 6 8 10
User 2 Input Events
Simulation 1 Event Order: 1 2 3 5 6 4 7 8 9 10
Simulation 2 Event Order: 1 2 4 3 6 7 8 10 9
5 (Packet lost)

(d) Latency, Jitter and Packet Loss

Fig. 3

The Effect on Multiple Distributed Simulations of Different Network Conditions.

The most straightforward network conditions for a peer-to-peer architecture to contend with are shown in Figure 3(a). With an update interval consistently less than the network delay, events can be delivered in the same order to all peers, and with suitable physical time-stamping, updates can be applied at the same time in all simulations.

Figure 3(b) illustrates the problem where latency is higher than the interval between successive updates. Where there is low jitter, adding a delay to the locally generated events (‘additional local lag’ [37]) can resynchronise event delivery so that it appears similar to Figure 3(a). This requires all input events, whether generated locally or remotely, to be delayed at each peer by an amount equal to the latency of the slowest network link in the system.

Figure 3(c) shows the effects of jitter. While the solution of injecting additional local lag might be reasonable under the conditions experienced in communicating between Manchester and The University of North Carolina (where there is low jitter and constant latency), in order to be fully effective, all events must be delayed by the *maximum* latency of the slowest link, a threshold much greater in the networks shown in Table I than the mean round-trip time. In the case of the Manchester–Amsterdam link, the total delay would need to be consistently 82ms, whereas the mean client–server round-trip would only be 27.6ms. Similarly in the Manchester–Bristol example a client–server architecture would experience a 14.3ms mean round-trip, but even the relatively low level of jitter would require all updates to be slowed to at least 44.1ms when using the additional local lag approach.

Figure 3(d) shows the problem that occurs when losing updates which, depending on protocol, may be impossible to detect. TCP/IP provides a guarantee of delivery, however lost packets still incur significant additional latency. Moreover because TCP/IP guarantees the order in which packets are delivered, following packet loss, subsequent packets in the stream will be queued until the lost packet is successfully delivered. UDP on the other hand provides a best-effort approach, delivering packets in the order in which they arrive but offers no notification of lost packets. This is particularly problematic where a single update may make a significant change in state to the simulation (for example connecting two objects together or triggering complex predefined behaviour). Additionally where updates only provide relative changes, the result of their incremental application may cause significant divergence.

Many existing peer-to-peer architectures work because of the relative simplicity of the behaviour they support. Parallel computation of complex behaviours such as multi-body simulations is a significant challenge, even to shared memory supercomputers, where communications latency is many orders of magnitude less than that on a LAN [38], [39].

C. Hybrid Architectures — Coping with Jitter and Packet Loss

It can be argued that since network technology exists that offers a relatively low-latency, low-jitter connection between the UK and USA (Figure 2(d)) that eventually all networks will evolve to that level. However, experience has shown that as capacity increases, applications evolve that utilise the extra bandwidth, such as Voice Over IP (VOIP), video conferencing and Network Attached Storage. While it is possible that increasing support for Quality of Service (QOS) mechanisms will reduce the problem of jitter at some point hence, it is likely that for the foreseeable future latency and jitter will continue to be a significant issue.

The performance of the network link to Labein in the afternoon shown in Figure 2(e) is typical of a highly congested network with both significant latency and jitter, and packets being dropped frequently. The high degree of jitter is likely to be caused by consecutive packets taking different routes from sender to receiver in order to avoid congested links, or by packets experiencing varying queueing times at different routers depending on other network traffic.

When the router queues get filled, packets are dropped, explaining why periods of peak latency coincide with packet loss. This would have a catastrophic effect on the synchronisation of a fully peer-to-peer system. These performance figures appeared to be characteristic of this particular link during the working day. However for comparison purposes the performance of the same link is shown at around midnight on the same day in Figure 2(f) where it behaves significantly better, demonstrating how the performance characteristics directly depend on network utilisation.

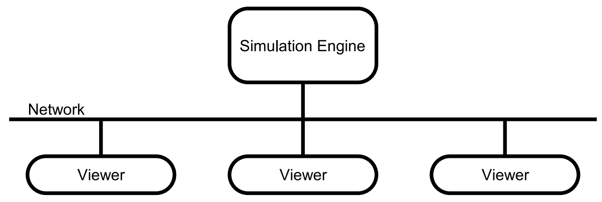
Given the dramatic change in network conditions that can occur, without quality-of-service guarantees peer-to-peer architectures do not degrade gracefully enough to support sophisticated behaviour models. Some of the assembly tasks we undertook using our virtual prototyping system (described later) took over an hour for an expert user to perform the entire procedure. Having the simulation catastrophically fail due to a sudden surge in network usage would not have been acceptable.

We have already shown that making peer-to-peer architectures robust against common levels of jitter introduces so much additional latency that it reduces the performance of the average case to well below that of a client–server solution; therefore if meaningful collaboration is to be successfully performed over slow and jittery networks, a hybrid solution must be found.

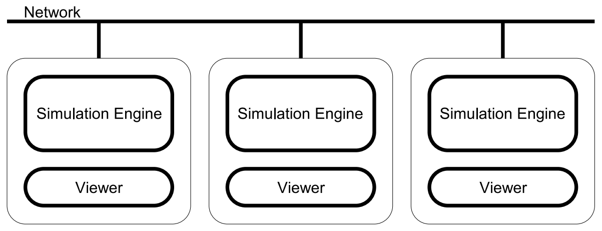
A third commonly used architecture that attempts to marry the low-latency advantages of a peer-to-peer network with the synchronisation benefits of a client–server architecture is to build a peer-to-peer system but then enforce object-based locking as illustrated in Figure 4(c) (an architecture occasionally called *token-ring* [18]). This eliminates the problem of out-of-order updates as only one user is able to interact with a given simulation object at once, however it also prevents simultaneous co-operation, and offers no way of scaling to support it.

For many applications this architecture will be sufficient, providing particularly good support for haptics, however it enforces rigid turn taking which is often unnatural. (While in most interactions, such as holding a conversation, social conventions result in turn-taking, people still expect to be able to interrupt each other.)

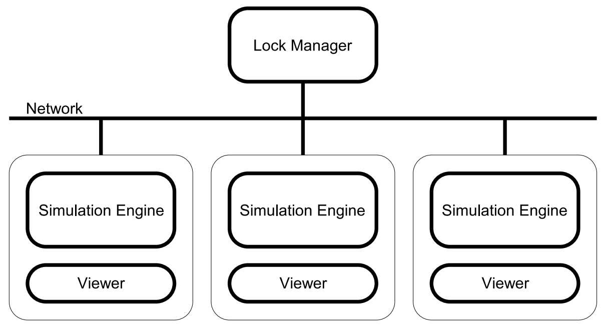
An improvement to this approach would be to provide the



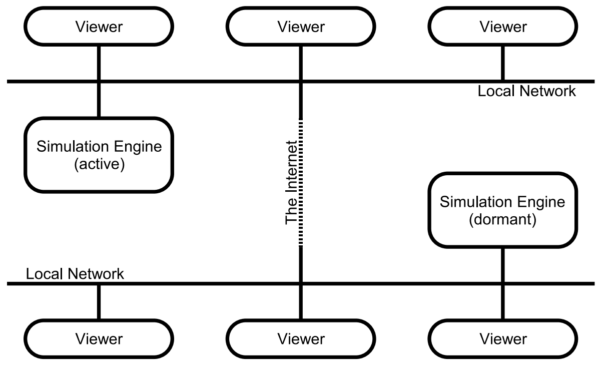
(a) Client–Server



(b) Peer-to-Peer



(c) Enforced Turn-Taking (Locking)



(d) Roaming-Server

Fig. 4
NETWORK ARCHITECTURES SUPPORTING COLLABORATION AND CO-OPERATION.

ability to identify and lock specific regions of a peer-to-peer network in which all connections between peers provide event delivery characteristics similar to those shown in Figure 3(a). An architecture similar to this was suggested by Hespanha *et al.* [19].

One problem with architectures such as these, however, is that it is not always intuitively clear to the users who is allowed to interact at any one moment in time. This can result in the ‘shared blanket’ problem where users on the periphery of a fast network-group are able to collaborate with all the other members except each other, and so either one joining the interaction locks out the other, while the rest of the group appears free to interact at will. Again, while this is a reasonable architecture for many applications to adopt, this is an additional restriction on the environment that has no real-world counterpart.

V. THE ROAMING-SERVER HYBRID ARCHITECTURE

In designing our distribution strategy, we aimed to avoid the possibility of simulations diverging yet still support low-latency interaction suitable for simultaneous co-operative haptic tasks. In order to achieve this, we exploit the observation that client–server architectures are fast enough to perform haptic rendering on local networks. In addition they perform more reliably than peer-to-peer architectures over slow or jittery connections for visual displays, while at the same time offering better support for complex rich behaviour.

In our architecture we utilise a number of servers, each containing simulation engines, and distributed across the network of participants. Typically there would be one server located on each local network as shown in Figure 4(d). One of these servers is also nominated to manage the administration of the environment itself and to ensure that logically only one of the simulation engines can be active at any time.

Initially all of the simulation engines are dormant. Clients connecting to the system profile the network link to each of these engines in order to determine their preferred server for subsequent interactions. When a user begins to manipulate an object, their client sends a request to the environment to activate its preferred server. If all the simulation engines are dormant, their request is granted, and the client is notified accordingly. All updates now occur between the client and active simulation engine in an identical manner to a standard client–server architecture.

If a second user wishes to either collaborate or co-operate with the first, their client issues a request to the environment for their preferred server to be made active. In this case the response returned rejects this demand and informs the requesting client of the existing active simulation engine. The client now uses this server in an identical manner to the previous user. Since both users are interacting with the same server they are free to interact with the same simulation entities, however depending on their network location the second user may suffer greater latency than the first.

The environment maintains a list of actively interacting users. If a short period elapses (two seconds in our implementation) with none of the users manipulating any entities, the simulation engines synchronise and the system enters its dormant state. The environment is now ready to activate whichever simulation engine is requested by the next user to interact. If during the two second period someone picks up or otherwise interacts with an entity then the synchronisation process is cancelled in order to maintain predictable behaviour.

Assembly sequences of the kind we are addressing here typically involve a small group of users assembling around one to two hundred components. In our particular test cases this takes over an hour to complete. During this time users frequently make context changes (selecting components, re-orienting their viewpoint and interacting with their user interface), and often pause to discuss the next stage of the assembly sequence. It is these natural breaks that afford us the opportunity to synchronise servers and roam if necessary.

An important benefit of the roaming-server architecture is that it provides an approach to building interactive applications that could be used to adapt many existing collaborative systems, both peer-to-peer and client–server. In our case, we chose to build our assembly and maintenance application, on top of our own Deva 3 [40]–[42] system. This offered the advantage that state migration is already well supported by the system [42]. However more sophisticated algorithms for state migration exist, with impressive performance figures being reported by the authors of the XEN virtual machine monitor [43].

It is important to note that this architecture avoids imposing additional programming complexity on the application designer and supports highly-iterative behaviours such as physical simulations particularly well. (In a peer-to-peer architecture these can easily diverge through small timing inconsistencies and rounding errors). While requiring no assumptions to be made about network protocols, the architecture will always guarantee that the simulation will be in a globally consistent state while at the same time adapting to usage patterns and network conditions.

A. *Virtual Assembly and Maintenance Simulation*

The DIVIPRO application is a system developed for virtual assembly and maintenance tasks, enabling collaborative and co-operative engineering design between geographically distributed design teams. It supports four distinct types of rich behaviour: collision detection, geometric constraints, flexible object simulation, and haptic-feedback. While collision detection enforces a minimal set of constraints on the motion of entities, most algorithms are unsuitable for simulating complex assemblies such as hinges and locating pegs in holes. Therefore geometric constraints are used to enforce these alignment restrictions. Additionally a form of 'snap dragging' model is useful in many virtual environments where spatial awareness and input devices are limited. Axial and planar geometric constraints activate based on a threshold relating the distance between the position and orientation of a component and potentially suitable axes. When a constraint activates, the component is 'snapped' the short distance to be correctly

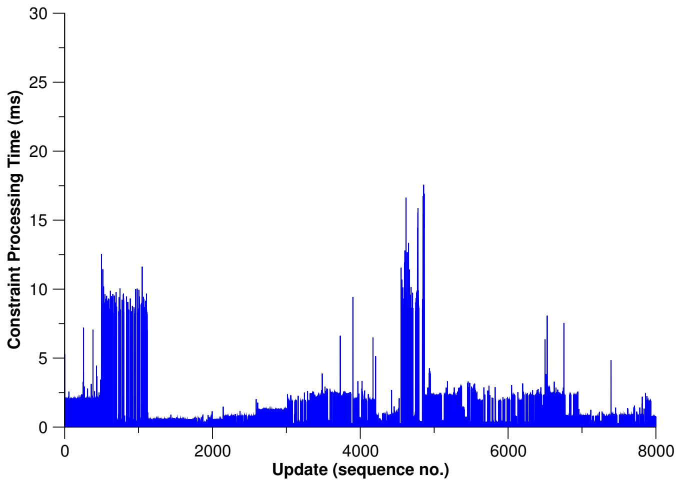


Fig. 5
LATENCY INTRODUCED BY THE SIMULATION ENGINE.

oriented and restricted to only move along the relevant axes unless a sufficient force is applied to break the constraint.

A large number of prototyping applications require some representation of flexible pipes and cabling harnesses. DIVIPRO supports a flexible object model that utilises a simulation specifically designed for simulating flexible pipes [44], [45]. This treats pipes as multi-body systems of connected masses and springs, and considers the distribution of mass and energy through the structure. External forces and environmental constraints such as collisions act on the body causing the model to compute a new equilibrium position. The model uses a closed-form analytical solution, integrating these equations of motion for the multi-body system by implicit time stepping methods in conjunction with a Newton-Raphson solver.

The computational complexity of collision detection, geometric constraint matching, and deformable pipe simulation varies depending on the geometric structure of the components being manipulated by the users. The graph shown in Figure 5 records the update latency introduced by the simulation engine for one of the sample assembly tasks, in this case the assembly of a blood-pressure monitor, part of which is shown in Figure 6. The model contains a large number of small parts which require careful assembly in a specific sequence. The two regions of highest processing time correspond to the manipulation and docking of complex components constructed from parametric surfaces including trimmed NURBS, such as the gear wheel shown near the bottom of Figure 6, which can potentially match a large number of axial constraints.

B. *The Deva 3 Implementation*

Deva 3 was originally designed as a client–server virtual reality architecture supporting a parallel cluster of machines for simulating the behaviour of the world. Related to this was the ability to migrate entities between server nodes both for load balancing purposes and to optimise the communication latency between interacting entities. As originally envisioned, all the server nodes were expected to be located physically

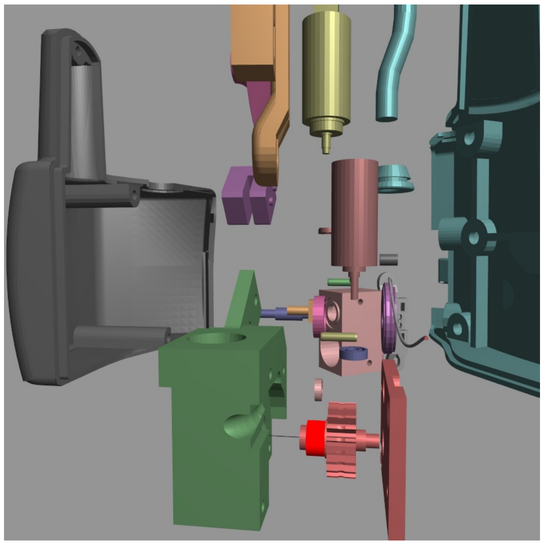


Fig. 6
THE BLOOD PRESSURE MONITOR TEST CASE.

close to each other on a very fast, high-bandwidth local network so as to avoid significant synchronisation problems within the server. Our initial approach was to build on this existing functionality and configure nodes in the parallel server to be distributed across the network. Using Deva's native load balancing it would be possible to migrate all of the entities on demand closer to the interacting user (hence the name 'roaming-server'). While this was possible for simple shared environments, the cost of initialising data-structures for collision detection, and automatically parsing the model's boundary representations in order to recognise potential constraints, proved too slow to undertake every time the locus of control changed. The solution was to replicate all simulation objects at each server node and to toggle the active server each time the locus of control moved.

C. *Exploiting Perception — Object and Subjects*

The core of the programming model adopted by Deva is that of a client–server system: in particular, at any given time the environment's semantic state is managed by a single node in the system, thereby avoiding the possibility of users seeing divergent behaviour. However the system provides for a more flexible approach to communicating changes in the environment to the clients.

Behaviour is characterised as either being *objective* (that is part of the synchronised, semantic state of the environment), or *subjective* (part of what is necessary to best-portray the correct interpretation of the objective state to the user). This distinction is similar to differentiating between the world around us, and our perception of it. Entities in the Deva environment are defined to have a single 'object' that resides on a server such as an instance of the multi-body simulator in the case of a flexible pipe. In order to either render or interact with an object, each client creates a corresponding 'subject'. The subject is responsible for the user's perception of the object: this includes both haptic and visual rendering, and communicating user interactions to the object. It is the responsibility of the object to decide when all the corresponding subjects in the system are to be updated. Likewise the information sent can be customised on a per-entity basis since both the object and subject can contain any arbitrary program code. In the case of the flexible pipes the object sends a list of control points to the subject which then extrudes and renders its own representation of the pipe locally. Both the objects and subjects can choose whether to use reliable messaging (usually to distribute semantically important events), or unreliable messaging (such as for streamed updates or unimportant background animation).

The system also implements a number of features to make the virtual environment more perceptually coherent in the face of network delays. In order to provide as high a frame rate as possible for visual displays, and to avoid glitches caused by jitter, updates can be delayed on arrival by the duration of the mean latency plus mean jitter. Locally these updates are then interpolated using cubic curves [46]. This is particularly suitable for users naturally taking turns while collaborating over slow or jittery links as it allows the passive observers to view smooth, accurate motion.

Subjects are also responsible for haptic rendering due to the requirement for a higher update rate than can be achieved with a remote server. Each haptic device is connected to a PC running a real-time daemon operating at 1KHz. This daemon is responsible for simultaneously applying a small set of physical constraints (such as preventing the penetration of planes and enforcing point, axial or planar geometric constraints).

The daemon also streams positional updates back to the simulation engine. State-related decisions, such as downloading and activating constraints in the daemons are still made by the active simulation engine, however the client's daemon enforces a particular constraint until it is instructed otherwise. This approach also has the added benefit of smoothing out the jitter caused by the simulation engine taking varying amounts of processing time depending on model complexity.

No workarounds are going to make simultaneous haptic cooperation effective where latency and jitter are particularly high. In this case the best that can be achieved is to disable haptically-rendered collision response, while continuing to use the device for high-resolution 3D input. Additionally in our particular application, the use of automatic geometric constraints to help align and orient components reduces the need for continuous high-resolution collision detection in order to successfully perform and demonstrate the assembly tasks. We also confirmed that alternative strategies such as using audio and visual cues [47], [48] to highlight constraints activating and collisions occurring were particularly effective when not using haptics, such as for desktop mouse users.

Given the lack of a human representation in our application, visual cues also proved effective for mitigating the lack of common social cues. Even though our test application allowed users to simultaneously manipulate the same object, they frequently *chose* to take turns due, in part, to the difficulty of knowing what other users were doing at any given time. (Similar behaviour has been reported in the CSCW literature. For example, in describing the relative merits of allowing simultaneous access in their text editing system, Ellis *et al.* report that while users initially interacted in a chaotic manner, they quickly established social conventions to mediate interaction resulting in surprisingly few conflicts [49].)

Our users initially attempted to synchronise their actions via voice communication. After making this observation we implemented a subtle background colour change when users were interacting, thereby providing an implicit contextual cue to enable social turn-taking where desired. This was also exploited to indicate to advanced users when the server was able to roam. Figure 7 shows two users collaborating. The user with a subtle green background (left image) is actively manipulating a flexible cable and cap into position onto a fuel injection control box while the participant with the pink background (right image) is passively viewing the task.

D. *The Roaming-Server in Use*

Figure 8 shows the total round-trip update latency experienced by two users performing a collaborative assembly sequence, separated by a network with similar performance to that shown in Figure 2(b). (This was simulated using the Linux 'netem' [50] queueing discipline.)

The application architecture is similar to that shown in Figure 4(d). The graph shows only round-trip times for actual packets sent, and hence 'dead-time', when the users are merely manipulating their viewpoints, is not shown. Spaces have been introduced into each individual user's sequence of packets so that they are shown correctly aligned to each other. Similarly the time taken for the simulation engines to synchronise is not shown (this takes around one to two seconds to complete).

User A begins the assembly sequence, inserting the axles and cogs of the blood-pressure monitor into its main housing. This causes their local simulation engine to become active and they are able to enjoy low-latency interaction. After completing this part of the assembly, they cease to interact, thereby relinquishing ownership of the simulation engine. The simulation engines synchronise their state at this point, and user B takes over to perform the next part of the sequence with their local simulation engine becoming active. Having attached the back panel and encoder wheel, they then pass control back to user A causing the active simulation engine to roam back to user A's local network. User A continues with their task for the remainder of the period shown, beginning with attaching the first half of the case. At this point user B assists user A by moving a component causing an obstruction. User B is forced to use user A's simulation engine and thus experiences a greater round-trip latency. However due to the local haptic rendering daemon maintaining the haptic update rate (and hence haptic stability), this communication delay is still within what we found to be generally acceptable levels for haptic collision response illustrated in Figure 1.

While this example would work reasonably well with a client–server configuration, one user would consistently suffer a slower response. In our architecture users only suffer these delays during co-operative manipulations. Under any of the conditions exhibited by the other wide-area networks profiled in Figure 2 it is unlikely that a client–server approach would

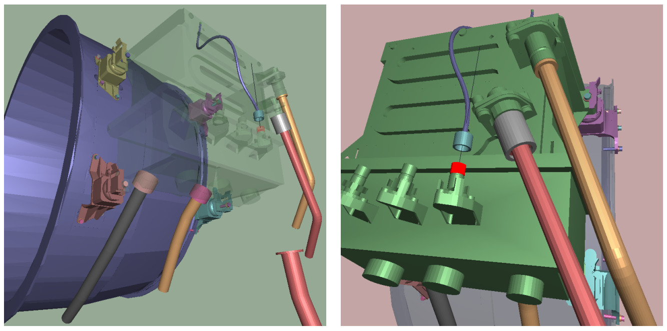
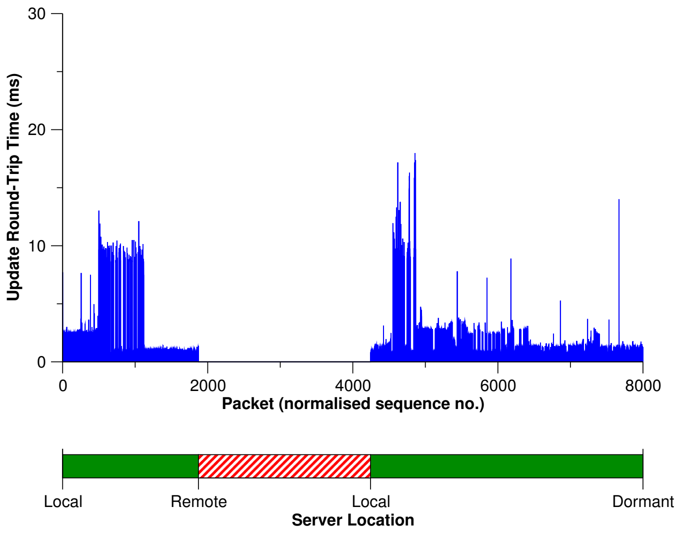
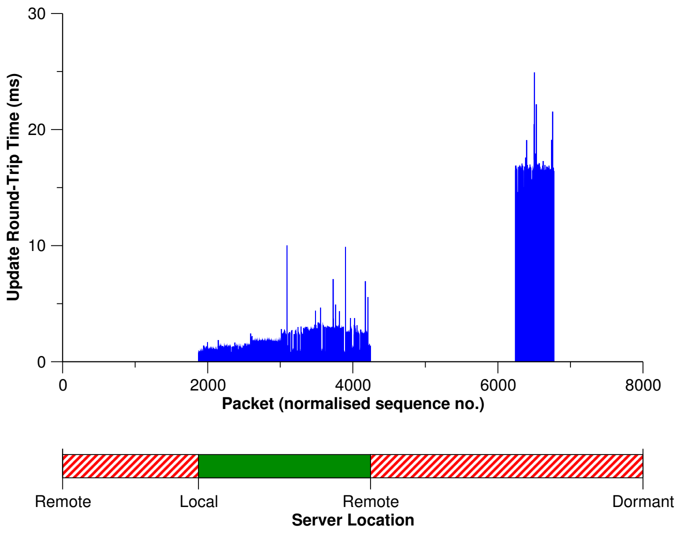


Fig. 7
TWO DIFFERENT USERS COLLABORATING ON AN ASSEMBLY TASK.



(a) User A



(b) User B

Fig. 8
ROUND-TRIP TIMES AND SERVER LOCATION FROM EACH USER'S PERSPECTIVE.

provide satisfactory haptic-interaction; however our approach will continue to support low-latency collaboration through turn-taking.

VI. Conclusion

We have shown how under common network conditions latency can be extremely problematic for interactive collaborative tasks in behaviourally rich environments, with neither client–server nor peer-to-peer architectures adequately addressing these issues. Many proposed network architectures concentrate on reducing latency, frequently preferring to adopt a peer-to-peer configuration for this purpose.

We measured the latency on a number of network links and found jitter to be a far more significant problem than the mean latency, with peak latencies many times the mean value. Consequently we argue that the belief that introducing additional delays into peer-to-peer systems will increase consistency is flawed. Consistency in a peer-to-peer system can only reliably be maintained when the additional latencies far exceed those of the mean round-trip time a client–server solution experiences.

We have described a hybrid distributed architecture for the effective management of complex virtual environments that operate over networks with significant latency and jitter using a novel roaming-server to maintain consistency. Our system fully supports haptic interaction, enabling users to feel as well as see the objects they are manipulating. Recognising that it would be impossible to meaningfully support simultaneous co-operation over networks where latency and jitter are particularly high, our architecture provides the best possible experience where a mixture of prevailing network conditions occur.

Users connected over slow links are not prevented from interacting simultaneously with each other, however their experience is likely to be less satisfying than if they chose a turn-taking strategy. At the same time, other users taking part in the same session are free to interact simultaneously with each other where network conditions permit. Most importantly, if people attempt to co-operate across poor links the system will not fail or enter an inconsistent state. Similarly at no point are any participants prevented from interacting with the environment, instead relying on natural, social conventions for mediation where desired.

The system was found to work well under a variety of network conditions spanning the full range of performance profiles shown in Figure 2 and Table I, both over simulated and real wide-area networks.

Acknowledgements

We would like to gratefully acknowledge the help of all our collaborators, in particular Teresa Gutierrez at Labein; Holger Weiss at DLR; Gunter Kadegge at KL Technik; Ernesto Arenaza at Sener; Bas Ruiter at FCS Control Systems; and Matt Aranha and Alan Chalmers at The University of Bristol. This work was funded under the grants EPSRC–GR/M66448 and EU–IST-200-28707.

References

[1] F. P. Brooks, Jr, "What's real about virtual reality," *IEEE Computer graphics and applications*, pp. 16–27, 1999.

[2] S. Jayaram, J. Vance, R. Gadh, U. Jayaram, and H. Srinivasan, "Assessment of VR technology and its applications to engineering problems," *Computing and Information Science in Engineering*, vol. 1, no. 1, pp. 72–83, March 2001.

[3] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 3, pp. 908–916, July 2003.

[4] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, San Diego, CA, 2003, pp. 92–101.

[5] Y. Liu, X. Liu, and E. Wu, "Real-time 3D fluid simulation on the GPU with complex obstacles," in *Pacific Graphics*, October 2004, pp. 247–256.

[6] C. Basdogan, C.-H. Ho, M. A. Shrinivasan, and M. Slater, "An experimental study on the role of touch in shared virtual environments," *ACM Transactions on Computer Human Interaction*, vol. 7, no. 4, pp. 443–460, December 2000.

[7] E. L. Sallnaes, K. Rassmus-Groehn, and C. Sjoestroem, "Supporting presence in collaborative environments by haptic force feedback," *ACM Transactions on Computer-Human Interaction*, vol. 7, no. 4, pp. 461–476, 2000.

[8] R. J. Adams, D. Klowden, and B. Hannaford, "Virtual training for a manual assembly task," *Haptics-e*, vol. 2, no. 2, October 2001.

[9] I. Oakley, S. Brewster, and P. Gray, "Can you feel the force? an investigation of haptic collaboration in shared editors," in *EuroHaptics*, 2001.

[10] M. O. Ernst and M. S. Banks, "Humans integrate visual and haptic information in a statistically optimal fashion," *Nature*, vol. 415, no. 1, pp. 429–433, January 2002.

[11] R. Komerska and C. Ware, "Haptic task constraints for 3D interaction," in *IEEE Symposium on Virtual Reality*, 2003.

[12] M. A. Otaduy and M. C. Lin, "Sensation preserving simplification for haptic rendering," in *ACM SIGGRAPH Transactions on Graphics*, vol. 22, San Diego, CA, 2003, pp. 543–553.

[13] M. A. Otaduy, N. Jain, A. Sud, and M. C. Lin, "Haptic rendering of interaction between textured models," in *SIGGRAPH: Sketches and Applications*, 2004.

[14] J. D. Hwang, M. D. Williams, and G. Niemeyer, "Toward event-based haptics: Rendering contact using open-loop force pulses," in *Twelveth International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS)*. IEEE Computer Society, March 2004, pp. 24–31.

[15] J. M. Hollerbach, E. Cohen, and W. Thompson, "Haptic interfacing for virtual prototying of mechanical CAD designs," in *DETC ASME Design Engineering Technical Conferences*, Sacramento, CA, September 1997, pp. 14–17.

[16] G. Zachmann and A. Rettig, "Natural and robust interaction in virtual assembly simulation," in *Eighth ISPE International Conference on Concurrent Engineering: Research and Applications*, Anaheim, CA, July 2001.

[17] G. C. Burdea, "Haptic feedback for virtual reality," in *Virtual Reality and Prototyping Workshop*, June 2003.

[18] P. Buttolo, R. Oboe, and B. Hannaford, "Architectures for shared haptic virtual environments," *Special Issue of Computers and Graphics*, 1997.

[19] J. P. Hespanha, M. McLaughlin, G. S. Sukhatme, M. Akbarian, R. Garg, and W. Zhu, "Haptic collaboration over the internet," in *The Fifth PHANTOM Users Group Workshop*, 2000.

[20] K. Montgomery, C. Bruyns, J. Brown, S. Sorkin, F. Mazzella, G. Thonier, A. Tellier, B. Lerman, and A. Menon, "Spring: A general framework for collaborative, real-time surgical simulation," in *Medicine Meets Virtual Reality*. Amsterdam: IOS Press, 2002.

[21] F. Bogsanyi and M. L. Miller, "Tool and object based synchronization in collaborative haptics," in *IEEE International Workshop on Haptic Audio and Visual Environments (HAVE)*, 2002, pp. 109–113.

[22] X. Shen, F. Bogsanyi, L. Ni, and N. D. Georganas, "A heterogeneous scalabale architecture for collaborative haptics environments," in *IEEE International Workshop on Haptic Audio and Visual Environments (HAVE)*, September 2003, pp. 113–118.

[23] M. Oliveira, J. Mortensen, J. Jordan, A. Steed, and M. Slater, "Considerations in the design of virtual environment systems: A case study," in *Second International Conference on Application and Development of Computer Games*, Hong Kong, January 2003.

[24] K. S. Park and R. V. Kenyon, "Effects of network characteristics on human performance in a collaborative virtual environment," in *IEEE Virtual Reality*. IEEE Computer Society, 1999, pp. 104–111.

[25] C. Gutwin, "The effects of network delays on group work in real-time groupware," in *European Conference on CSCW*, Bonn, 2001, pp. 299–318.

[26] G. C. Burdea, *Force and Touch Feedback for Virtual Reality*. John Wiley and Sons, Inc., 1996.

[27] R. Q. van der Linde, P. Lammertse, E. Frederiksen, and B. Ruiter, "The HapticMaster, a new high-performance haptic interface," in *Eurohaptics*, 2002, pp. 1–5.

[28] S. Singhal and M. Zyda, *Networked Virtual Environments Design and Implementation*. ACM Press SIGGRAPH Series Addison Wesley, 1999.

[29] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: A network software architecture for large-scale ves," *Presence Teleoperators and Virtual Environments*, vol. 3, no. 4, pp. 265–287, 1994.

[30] S. Benford, J. Bowers, L. E. Fahlén, and C. Greenhalgh, "Managing mutual awareness in collaborative virtual environments," in *Virtual Reality Software and Technology*, Singapore, 1994, pp. 223–236.

[31] C. Greenhalgh and S. Benford, "MASSIVE: A collaborative virtual environment for teleconferencing," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 2, no. 3, pp. 239–261, September 1995.

[32] O. Hagsand, "Interactive multiuser VEs in the DIVE system," *IEEE Multimedia*, vol. 3, no. 1, pp. 30–39, 1996.

[33] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[34] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, 1985.

[35] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and timewarp: Providing consistency for replicated continuous applications," *IEEE Transactions on Multimedia*, vol. 6, no. 1, pp. 47–57, February 2004.

[36] B. Mirtich, "Timewarp rigid body simulation," in *SIGGRAPH Computer Graphics*, New Orleans, LA, 2000, pp. 193–200.

[37] M. Mauve, "Consistency in replicated continuous interactive media," in *ACM Conference on Computer Supported Cooperative Work (CSCW)*, Philadelphia, PA, December 2000, pp. 181–190.

[38] R. Cozot, "From multibody systems modeling to distributed real-time simulation," in *Twenty Nineth Annual Simulation Symposium*, New Orleans, LA, 1996, pp. 234–241.

[39] J. G. Cleary, M. Pearson, and H. Kinawi, "The architecture of an optimistic CPU: the WarpEngine," in *Twenty-Eighth Hawaii International Conference on System Sciences*, vol. 1, Wailea, HI, January 1995, pp. 163–172.

[40] S. R. Pettifer, "An operating environment for large scale virtual reality," Ph.D. dissertation, The University of Manchester, 1999.

[41] S. Pettifer, J. Cook, J. Marsh, and A. West, "DEVA3: Architecture for a large-scale distributed virtual reality system," in *ACM Symposium on Virtual Reality Software and Technology*, Seoul, Korea, 2000, pp. 33–40.

[42] J. Marsh, "A software architecture for interactive multiuser visualisation," Ph.D. dissertation, The University of Manchester, 2002.

[43] C. Clark, K. Fraser, S. Hand, J. G. Hanseny, E. July, C. Limpach, I. Pratt, and A. Wareld, "Live migration of virtual machines," in *2nd Symposium on Networked Systems Design and Implementation*, Boston, Massachusetts, 2005.

[44] H. Weiß, "Dynamics of geometrically nonlinear rods: I. mechanical models and equations of motion," *Nonlinear Dynamics*, vol. 30, pp. 357–381, 2002.

[45] ——, "Dynamics of geometrically nonlinear rods: II. mechanical models and equations of motion," *Nonlinear Dynamics*, vol. 30, pp. 383–415, 2002.

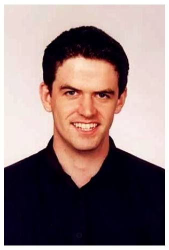
[46] J. Marsh, S. Pettifer, and A. West, "A technique for maintaining continuity of experience in networked virtual environments," in *UKVRSIG*, Sept. 1999.

[47] M. Fraser, T. Glover, I. Vaghi, S. Benford, C. Greenhalgh, J. Hindmarsh, and C. Heath, "Collaborative virtual environments," in *Third International Conference on Collaborative Virtual Environments*, San Francisco, CA, 2000, pp. 29–37.

[48] R. Sekuler, A. B. Sekuler, and R. Lau, "Sound alters visual motion perception." *Nature*, vol. 385, p. 308, January 1997.

[49] C. A. Ellis, S. J. Gibbs, and G. Rein, "Groupware: Some issues and experiences," *Communications of the ACM*, vol. 34, no. 1, pp. 39–58, January 1991.

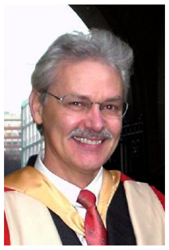
[50] S. Hemminger, "Network emulator," Online Document, March 2005, ⟨http://developer.osdl.org/shemminger/netem/⟩.



James Marsh obtained a first class honours BSc. degree in computer science in 1997, an MRes. in Informatics in 1999, and a PhD in 2003, studying software architectures for collaborative visualisation. He is a post-doctoral research associate in the Advanced Interfaces Group at The University of Manchester, United Kingdom. His main research interests are in the areas of collaborative virtual environments, distributed systems, and interactive scientific visualisation. He can be contacted via email at: james.marsh@manchester.ac.uk.

Mashhuda Glencross obtained her first degree in polymer science in 1992, her MSc. in computer science in 1994, and PhD in 2000, studying interactive physically-based modelling. She is a post-doctoral research associate in the Advanced Interfaces Group at The University of Manchester, United Kingdom. Her research interests span the areas of physically-based modelling, collaborative virtual environments and haptics. She can be contacted via email at: mashhuda.glencross@manchester.ac.uk.

Steve Pettifer is a lecturer in the School of Computer Science at The University of Manchester. His PhD was awarded on the subject of distributed virtual reality in 1999. His research interests include collaborative systems, abstract and scientific visualisation, human computer interaction and distributed computing. He can be contacted via email at: steve.pettifer@manchester.ac.uk.



Roger Hubbold is Professor of Virtual Environments in The School of Computer Science at The University of Manchester, where he heads the Advanced Interfaces Group. He holds a first class BSc. in Engineering (1967) and a PhD. in Computer Graphics (1971). His research interests include collaborative virtual environments, haptics, visualisation and virtual environment software architectures and algorithms, computer vision techniques for unencumbered interaction, and construction of virtual environments from images and video. He has published over 70 peer-reviewed papers and books in this field. He is a Fellow of Eurographics and an Associate member of IEEE and ACM. He can be contacted via email at: roger.hubbold@manchester.ac.uk.