Network Infrastructure for Massively Distributed Games

Daniel Bauer
dnb@zurich.ibm.com

Sean Rooney
sro@zurich.ibm.com

Paolo Scotton
psc@zurich.ibm.com

IBM Research
Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon, Switzerland

ABSTRACT

The popularity of hypertext documents led to the need for specific network infrastructure elements such as HTML caches, URL-based switches, web-server farms, and as a result created several new industries as companies rushed to fill that need. We contend that massive distributed games will have a similar impact on the Internet and will require similar dedicated support. This paper outlines some initial work on prototyping such support. Our approach is to combine high-level game specific logic and low-level network awareness in a single network-based computation platform that we call a *booster box*.

Keywords

network infrastructure, massively distributed games, network processors

1. INTRODUCTION

The amount of information exchanged in a multi-party communication session grows with the square of the number of participants. This means that such sessions require special techniques if they are to scale to large communities of users. These techniques include caching, aggregating, filtering, and intelligent forwarding; for example, some participants may be only interested in information from certain other participants so they need only to receive a subset of the information transmitted during the session.

These techniques are all to a lesser or greater degree application-specific, meaning that in general they are implemented in software on a server at the edge of the network. This has two drawbacks: first, all the traffic must cross the network from the clients to the server, resulting in unnecessary load on the network and server; second, the server being remotely located has at best only a very approximate view of the network state and therefore cannot take network state into account when applying these techniques.

Netgames 2002 April 16-17, 2002, Braunschweig, Germany.

We propose a different approach in which some of the server functions are executed on computation platforms — *booster boxes* — which are co-located with routers and are aware of the state of the network in their vicinity. Booster boxes can perform application-specific functions in the network, reducing the load on the servers. We argue that in this way some classes of applications, for example massive multi-player on-line games, which currently are unfeasible for large numbers of participants become possible.

Although distributed games are only one example of an application type that can run on such a platform, they are particularly interesting as they can potentially generate a sufficiently large revenue stream to make it worthwhile for the Internet service provider (ISP) deploying them[1]. How large a market there is for such games depends on human behavior and is therefore a social rather than a technical question. However, the gain in popularity of networked games and the increasing ease of access to the Internet will certainly dramatically increase the number of participants in on-line games. We anticipate multi-user games with millions of participants.

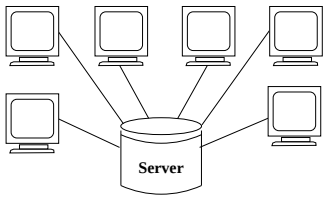
We foresee that bandwidth to the end-user will increase significantly, and that broadband access will become commonplace because of technologies such as ADSL and cable modems. Asymmetric bandwidth allocation, in which more bandwidth is available downstream than upstream, is an ideal match for large multi-user games, where users typically receive more data than they produce. Additionally, we assume that bandwidth at the server side is not a constraint. However, servers (or server farms) running the game have to handle an extraordinarily large number of events per time unit. The main limiting factor in supporting such games is server resources such as bus I/O and processing capabilities, i.e. CPU cycles and memory access.

The paper is organized as follows: first we describe the motivation for such an approach in more detail, then we show how this approach can be implemented in a booster box, next we outline the architecture of such a booster box, and finally we describe the booster box game support we are currently implementing as a proof of concept.

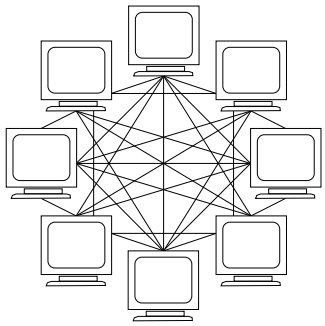
[1]The European ISP Wanadoo started in January 2002 a massive multi-player game on a subscription basis of ten euros per month. Their financial report notes that 70% of adherents also possess fast bandwidth connectivity, suggesting that Wanadoo sees their network game as a "killer application" for ADSL [21]

2. WHY COMPLEX GAMES NEED NETWORK SUPPORT

Clients of networked games periodically emit events [2] to be received by some subset of other clients. Figure 1 shows the two basic models. In the client-server approach, clients send events to a server, which then decides which other clients should receive that possibly interpreted event. In the peer-to-peer model, clients send events to all other clients, which then locally determine whether the event is of interest and how to interpret it.



(a) Client-server



(b) Peer-to-peer

Figure 1: Event-distribution model

2.1 Client-Server

In the client-server approach, a central server or a central server farm processes all events from the clients. In order to have an idea of an upper bound for how many game events a server can handle per second, we take an HTTP server as a reference. Although HTTP traffic and game traffic are very different, the behavior of an HTTP server is less complex and therefore can be used to calculate an upper bound. Game servers in general are more complex because information sent by one client must be correlated against that sent by others; this complexity typically increases with the square of the number of clients. Moreover, whereas each HTTP request sent corresponds to one HTTP reply, in a game server a given event sent by a client may be forwarded to many other clients. Finally, for commercial reasons industry solutions tend to be optimized for handling HTTP requests.

Regnier [15] bench-marked an Apache HTTP server running on a Linux PC with a Pentium III 800 MHz processor and a 64 bit 33 MHz bus connected to a Gigabit Ethernet as being able to handle approximately 2000 HTTP transactions per second, each transaction having a size of 256 bytes. Obviously this architecture is not representative of large server farms, such as those used to host the Olympic games' website, that involve hundreds of clustered workstations and front-ended with intelligent load balancers. However, the maximum reported load handled by these servers is on the same order of magnitude [7]. Whereas network bandwidth is abundant, the bottlenecks of such systems are CPU cycles, memory bandwidth, and server I/O.

We conclude that server farms handling loads greater than 10^5 HTTP requests per second currently are infeasible with existing technology and, by extension, so are million-person games requiring as little as a single event per minute. Smet *et al.* [17] report a required latency of 500 ms for strategy games, and 100 ms for games involving hand-eye motor control; these are clearly inconsistent with a system that can handle only an event per minute from each client.

2.2 Peer-to-Peer

The peer-to-peer topology is often used in games where the number of participants is small. The main advantages are low latency, as messages are not relayed by a server, and robustness, as there is no single point of failure. The main disadvantage is the lack of scalability. If every event is sent to every client, then each client is equivalent to a central server, with the difference that the client does not have to forward the event any further but only filter those events that are of interest and interpret them appropriately. In the preceding section we concluded that even large server farms are not capable of handling loads involving millions of participants; the total amount of traffic sent in a peer-to-peer model grows with the square of the number of clients.

Real-time strategy games constitute a special case in which each player controls a large number of game entities. Distributing state information of each entity clearly limits scalability, as described in [6]. Therefore, the *Age of Empires* series takes a different approach in which each peer runs the entire simulation and distributes the user's input to all other peers. Consequently, all peers execute the same commands at the same time and thus remain consistent. This solution significantly reduces the number of events distributed among the peers. However, scalability is limited by the computational power of the weakest peer, because each peer needs to run the complete simulation.

In conclusion, neither a pure peer-to-peer nor a client-server architecture is adequate to support a million-person real-time game. Porting some of the application's intelligence into the network – implementing some kind of *hybrid* approach – will overcome these scalability problems.

3. DELEGATING SOME APPLICATION FUNCTIONS TO THE NETWORK

If the server farm itself is distributed, such that only some subset of clients is handled by each part of the server farm, then we attain the desired level of performance. However, servers still need to coordinate states among themselves. If they were to do this naively, e.g. by sending every event they received to all other servers, then each server would receive all events and the situation would not be any better than in the completely centralized solution.

[2]Game events typically describe changes of state, such as a figure moving in a virtual environment.

Servers need to reduce the amount of information they forward to other servers by only sending relevant information. For example, a given event should only be forwarded to a server if there is at least one interested client registered with that server. Let us assume that there is no correlation between interest in an event and physical location. Then, as the number of clients rises, the probability that there is at least one client registered with a server interested in a given event approaches unity for a fixed number of servers.

The correlation overhead of the distributed-server approach limits its scalability. This overhead can be significantly reduced by enhancing the network with coordination, computation, and storage services that are provided by *network-aware* booster boxes, as shown in Figure 2.

Booster boxes acquire network awareness by monitoring traffic, measuring network parameters such as delay, by participating in routing exchanges, or by acting as an application-layer proxy server. Each booster box serves a number of clients in its network vicinity and performs caching, filtering, forwarding and redirecting of game events in a game-specific way. Compared to distributing the servers across the network as proposed in [8], the main advantage of this approach is that booster boxes combine network and application awareness in a single entity. For example, forwarding decisions can be based on high-level knowledge, e.g. the sender's location in the virtual space, as well as on network-level knowledge, e.g. current network delay.

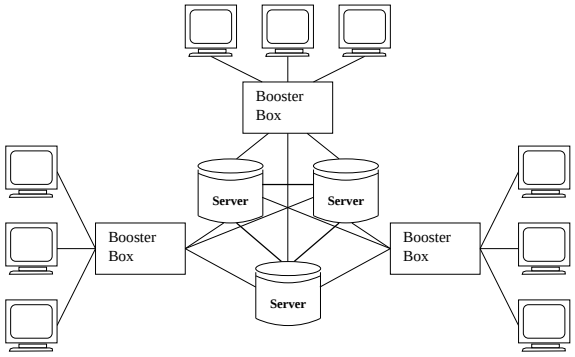


Figure 2: Distributed servers with booster boxes

We envisage that booster boxes will be deployed at the edges of an ISP network, attached directly to the IPS's access routers. A schematic overview of the deployment is given in Figure 3. These booster boxes might be considered as members of the broad range of appliances that the IETF terms "middleboxes"[18]. The ISP will allow privileged third parties to instrument their booster boxes in a way specific to the applications they wish to support.

A piece of application-specific code, called *booster*, is executed on the booster box. The booster box provides interfaces that allow the booster to observe and manipulate data streams, and to participate in control and management protocol operations. Boosters use the following fundamental operations to reduce the load on the servers:

Caching. Boosters cache non-real-time information and answer on behalf of the server.

Aggregation. Events from two or more clients might be aggregated. In the simplest case, several redundant events arrive within a given time frame, and only one is forwarded to the server. In other cases, the booster computes an aggregated event by performing a function that otherwise would have been performed on the server.

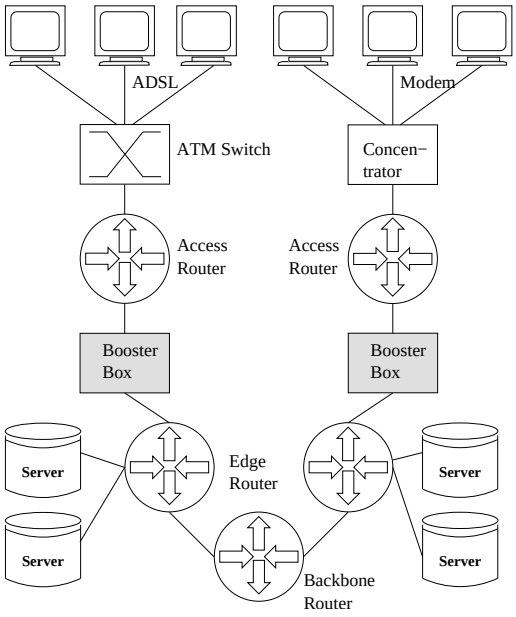


Figure 3: Deployment of booster boxes

Intelligent filtering. Depending on the state of the game, events may no longer be relevant. These events can simply be dropped by the booster.

Application-level routing. A game supporting millions of participants will be implemented on distributed servers. The booster acts as an application-level router, and sends an event only to those servers that are responsible for handling it.

4. ARCHITECTURE

The booster box architecture is divided into a *booster layer*, in which the high-level logic resides, and a *data layer*, which actually does the packet forwarding. Figure 4 gives an overview of the booster and the data layers.

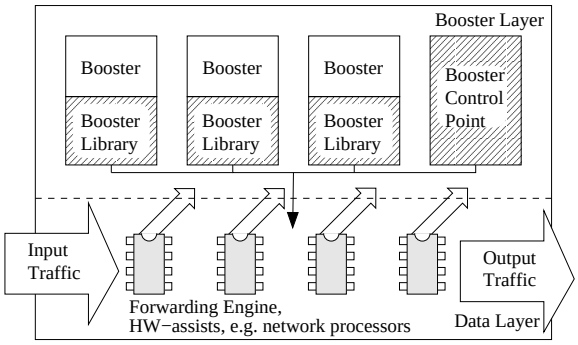


Figure 4: Booster- and data-layer overview

4.1 Data Layer

As shown in Figure 3, booster boxes are positioned between access and edge routers. For the bulk of traffic, booster boxes behave like ordinary layer-2 forwarding devices, e.g. like ethernet switches. Thus, the forwarding function in the booster boxes' data layer is kept very simple, because no forwarding tables have to be maintained. Besides forwarding, the data layer also copies or diverts selected traffic to the booster layer (see Figure 5). The description of which traffic to copy or divert is specific to each booster running in the booster layer. It is to be expected that only a small fraction of the overall traffic is copied or diverted, and that booster operations are applied to this small fraction only.

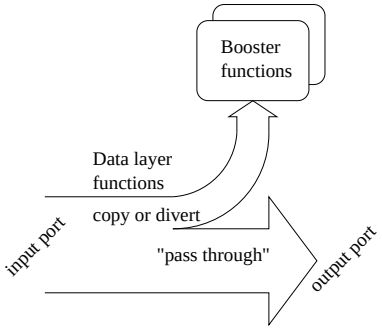


Figure 5: Booster-box traffic forwarding

The application programming interface (API) that boosters use to specify the traffic to copy or divert is very simple. We decided to use the packet descriptor format of the Unix packet capture module *libpcap* as the standard format for specifying which packets to copy. Similarly, we use the Linux *iptables* packet filter format for diversion. However, the format of the expression languages does not imply any particular implementation for these functions.

The data layer must be able to handle packet forwarding at speeds equivalent to the port of a residential access router, i.e. in the range of 155 Mbit/s to 1 Gbit/s. At the same time it must be able to process the copy and divert filters, and test packet headers against them.

A pure software solution is not able to handle such line speeds, whereas a pure hardware solution does not offer sufficient flexibility. Our approach is to use network processors for implementing the data layer of the booster box. Although the term network processor (NP) covers a wide variety of processors with different capabilities and designed for different markets — an excellent overview can be found in [16] — the simplest way to think of a NP is as a general-purpose processor with access to many network-specific co-processors, performing tasks such as checksum generation, table look-up, and header comparison. Arbitrary network-forwarding code can be written in a high-level language such as C (augmented with pragmas for co-processor invocation) compiled and loaded into such a processor. The NP therefore is a mid-point between a pure hardware and pure software solution.

The data-layer API provides an abstraction of the actual copy and divert mechanisms. Implementation details are shielded from the boosters. Booster use the same primitives independently of the underlying implementation, e.g. Linux kernel, NP, or dedicated hardware. Note that there is no functional difference between packet forwarding using an NP and, say, a Linux kernel, the only difference being speed. In our initial prototype we use both a Linux kernel and the NP, switching between them for different applications. The main advantage of using Linux is that built-in functions exist already, e.g. for packet diversion, which on an NP would have to be written manually. Some of our experience with implementing functions using a NP is given in Section 5.1.

4.2 Booster Layer

The booster layer consists of a set of boosters and a control point that is common to all boosters. Each booster is a combination of application-specific logic and generic booster library functions. Boosters can be executed either on the general-purpose CPU, or on the network processors or some combination of the two. The application specific code is trusted, being either written by the ISPs themselves or supplied by trusted third parties. The library contains the API through which the boosters can call the data-layer operations described above. The boosters execute independently, however certain operations need to be coordinated. The booster control point coordinates the boosters, and manages any data that is common to all of them. For example, when a booster requests a traffic-diversion operation, a message is sent to the control point by the code in the generic library that implements that operation in order to ensure that no conflicts occur.

In addition, the control points on different booster boxes build a QoS-aware overlay network between booster boxes, which boosters may use to transport data. This overlay network provides routing functions for low-delay and high-throughput traffic. The rationale behind the booster overlay network (BON) is that QoS mechanisms are not widely deployed in today's Internet, and never in combination with advanced multicast support. Numerous games, however, require low and stable latency. In today's Internet, routing information exchanged between providers is heavily aggregated and filtered by the border gateway protocol (BGP), mainly for scalability and policy reasons. Together with BGP's route-dampening algorithm, which is used to prevent route flapping, route convergence times in the range of tens of minutes in the case of errors occur rather frequently [11]. Clearly, such a network behavior inhibits a smooth operation of networked multi-player games. A BON is able to circumvent these problems. The key idea is to use a scalable link-state routing approach in contrast to BGP's path-vector mechanism. The link-state approach allows the computation of redundant paths between node pairs, whereas the path-vector method only advertises a single path between nodes. Thus, link-state routing allows a fast switching of paths in the case of errors. In addition, the multicast capability ensures that a large number of participants is able to receive data quasi simultaneously.

A single BON on top of the IP network layer is maintained for all applications supported by the booster box. The communication endpoints of the BON, however, differ for each application. Each booster registers endpoints individually. This results in a BON in which the "base" topology is common to all applications, but the endpoints are not. An overview of the BON structure is given in Figure 6. It shows that the BON (middle) is a connected subgraph of the IP topology (bottom). Different applications advertise different reachable endpoints, as indicated by the triangles

(top).

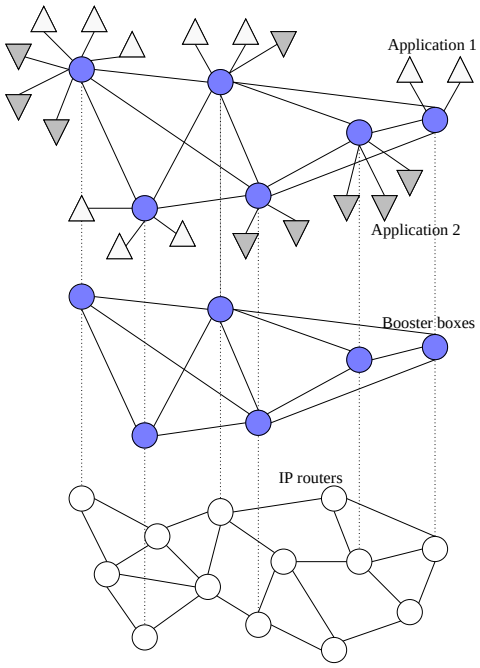


Figure 6: Booster overlay networking overview

The BON is maintained by a hierarchical link-state routing protocol, which is designed along the lines of the ATM control protocol PNNI [5]. Only a subset of PNNI's features is needed, however, thus the BON is considerably simpler than PNNI. The main differences to PNNI are the lack of a signaling protocol, as no resource reservation is done and as forwarding is done on a hop-by-hop basis, using time-to-live counters to prevent loops. Also, the link metrics are measured values of the available bandwidth and the expected delay. BON supports only two traffic classes, a low-latency and a high-throughput class. For each class, a distinct forwarding table is computed. Furthermore, the protocol uses an auto-configuration mechanism that results in a regular routing hierarchy.

BON uses its own addressing scheme, which facilitates address aggregation in hierarchical routing. Addresses are composed by prepending prefixes to the existing IP addresses. Data that is forwarded is encapsulated in a BON header. This header includes BON addresses for source and destination as well as a field defining the routing policy, i.e. low latency or high throughput. This policy field is consulted in each hop, and defines the forwarding table that is used. BON packets are encapsulated in IP when forwarded between booster boxes. On the last hop, both the outer IP header as well as the BON header are removed.

The auto-configuration groups booster boxes that are "network close". Booster boxes use three criteria when judging "network closeness" to a potential peer: hop count, bandwidth, and latency. These parameters are measured using a combination of the *pathchar* [10] and *packet tailgating* [12] techniques. Both peers apply measurements to each other to allow for different routes to be taken on the forward and return path.

These measurements techniques are slow as they require the transmission of a large number of probes to be sent. They are suitable for establishing link-state adjacencies, but not for measuring transitive link metrics, for which we use Network Weather Forecasting (NWS) [22]. NWS allows the future network behavior between two points to be *predicted*; it assumes that an NWS sensor is being run on each of these points. It periodically measures the TCP connection set-up time and round-trip time (RTT) for various sizes of TCP segments between these points. It then applies a set of statistical techniques to estimate the network behavior. Estimated values are continually compared with measured values to refine the prediction model.

For related work regarding overlay networks see [4] and the references therein.

5. USING THE BOOSTER BOX

In this section we motivate the general architecture described in Section 4 with some examples of how such a general-purpose network-computation platform can be used to assist in scaling various applications. Our approach has been to develop these applications in parallel with the development of the booster box itself in order to test its adequacy as well as to demonstrate its usefulness.

5.1 Example: Large Interactive Game Show

In this scenario questions are broadcast to a large number of spectators using the normal television network. The spectators can participate in the game by sending replies to the television station's server over the Internet. Users that answer incorrectly are removed from the game.

A centralized approach requires the television station's server to handle tens of millions of replies within a short time period, i.e. the maximum period in which a user is allowed to answer. By using both intelligent filtering and the aggregation functions in the booster box, the total load at the television station's server can be reduced exponentially.

If the booster box only forwards correct replies to the server, then the total traffic the server is required to handle is reduced by a factor that is inversely proportional to the probability that a user answers correctly; a parameter that to a great extent is under the station's control.

If the booster box combines all correct answers received within a given time window into a single packet containing all the corresponding user identifiers, then the total traffic the server is required to handle is proportional to $1/n^a$, where n is the average number of packets combined in a time window and a is the average number of booster boxes across which the answer is propagated.

We developed this, admittedly somewhat artificial, application to gain experience with implementing functions on an NP. We choose to use IBM NP4GS3 network processor [14].

Figure 7 shows the general architecture of the IBM NP4GS3. The NP basically consists of line interfaces, a set of pico-processors (EPC), and an embedded PowerPC processor. Packets are processed at line-speed by the pico-processors[3], which are programmed with a low-level language called *pico-code*. Control and management operations are implemented in an external controller that runs either on a separate Linux PC or on the embedded PowerPC. The controller can also be used to execute sophisticated operations on packets that do not require to be handled at line speed. The controller

[3]A scheduling mechanism assigns an incoming packet to the first pico-processor available.

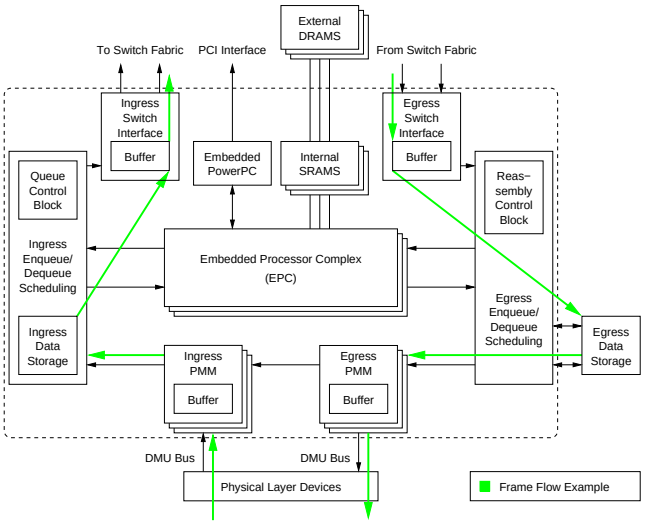


Figure 7: General architecture of the IBM NP4GS3

software communicates with the NP using a standard socket-based interface.

Our initial thought was to implement the entire game-show application as a piece of pico-code and load it into the NP; however the internal memory on the NP for saving state information is limited. Consequently, the number of packets that can be combined is restricted. We therefore decided to adopt a hybrid approach. The NP filters packets destined to the game server and checks for correct answers. Packets with incorrect answers are dropped, the others are forwarded to a process running on the external Linux PC. This process receives the packets and combines them until either a maximum number is reached or a timer expires. It then sends the aggregated information to the server.

The implementation in the NP consists of roughly 20 lines of pico-code, adding an additional latency of 300 ns to the packets which do not belong to the game show. Nevertheless, these packets are handled at line speed. In contrast, the aggregation process does not operate at line speed. However, it fully meets the real-time requirements of the game.

5.2 Example: Game with Large Virtual Space

In this scenario the game takes place in a virtual space. Each user controls an avatar that moves freely in this space. Movements of the avatars are encoded in events that are sent to a server. Events arriving from different clients are correlated by the server, which computes a modified state of the game. This state information is then sent back to the clients. This basic model is used by role-playing games such as EverQuest [1]. Although EverQuest claims that tens of thousands of people can participate simultaneously, a user is in fact restrained to choose one of about 40 servers on which to be located. As it is not possible to move between servers, each server is in fact independent. Therefore, EverQuest is better thought of as 40 independent instances of the same game, each of which handles about 2000 players.

By using application-level routing, a more flexible and scalable game can be achieved. In this approach, the virtual space is divided into zones. Each zone is handled by a specific server. Instead of clients connecting directly to a server, they connect to the closest booster box. The booster boxes forward events to the server in charge of the zone in which the player's avatar is located. The servers dispatch the new game state to the booster boxes, which forward it only to the interested clients.

When users move between zones, the booster box dynamically changes the server to which events are sent without users being aware. A booster box may also choose to send events or summaries of events to more than one server, so that for example the existence of a new player would be transmitted to all servers to permit a quick transfer between servers if that player chooses to move between zones.

By creating an indirection between clients and servers, booster boxes allow much larger continuous virtual spaces than are currently commercially available. As by definition the virtual location appears the same to all users, the servers can periodically send a single view to the booster boxes, which can determine which users should receive that view, creating a simple application-layer multicast distribution tree. Figure 8 shows such a configuration in which players may be at one of two virtual locations: one represented shaded the other not. A player in the "light" location sends an event to its local booster box, which then dispatches it to the appropriate server. The server updates the game state, and dispatches it to all booster boxes, which replay it to the appropriate players.

Our initial thoughts are that a variety of different games use the notion of virtual location but each game encodes this information in its own way. It is possible to write game-specific boosters which extract the location information in order to determine which server to send the event to. However, a more efficient solution is to create a game transport protocol in which the virtual location is encoded independently of the particular game. This has advantage that a game booster could to a large extent be independent of the game.

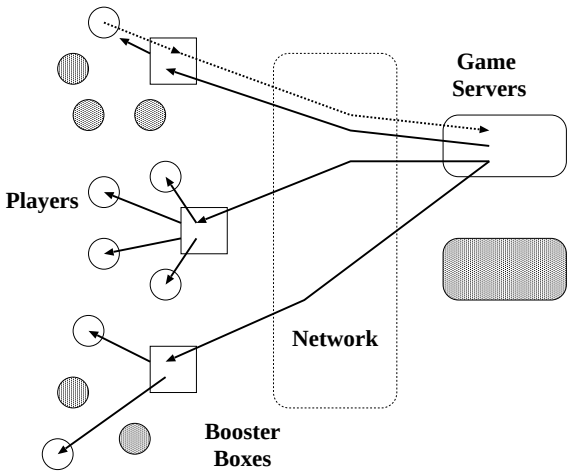


Figure 8: Game with large virtual space

5.3 Other Applications

We believe that on-line distributed games are particularly amenable to be "boosted", as shown in the preceding sections. In this section we quickly outline some other applications that are not game-related but can benefit from the booster-box approach.

5.3.1 Peer-to-Peer File Sharing

Gnutella [9] is an example of a file-sharing protocol that does not require a central server. Such approaches are very resilient to failure and have other advantages such as the number of locations at which a file can be found is proportional to the file's popularity. However, they are not very scalable as a single search operation results in the flooding of a large number of request messages.

Current work investigates making peer-to-peer file-sharing protocols more scalable by using boosters. Boosters can reduce the amount of traffic by caching, aggregating requests, and performing intelligent forwarding.

5.3.2 Floating Car Data

Floating Car Data (FCD) [3] is a label for a set of different initiatives that involve gathering sensor information from cars, processing it, and using this to generate useful information such as traffic-flow predictions. Sensor devices are placed in the cars to measure parameters such as speed or location information. The resulting data is transmitted in real time to back-end servers, first using General Packet Radio Service (GPRS) and later third-generation wireless communication and/or wireless LAN technologies. Potentially a huge amount of data can be produced by each car. Current work looks at how boosters can aggregate such information, e.g. by sending a single message saying that fifty cars are traveling at 40km/h, rather than sending fifty separate messages.

6. RELATED WORK

An excellent overview of the problems arising with the development of networked multi-player computer games can be found in [17]. Smed *et al.* distinguish four major areas effected by multi-player on-line games, which are all addressed by the booster box, as shown below:

Networking resources. The use of network resources is reduced by processing information at an early stage or by distributing data across multiple servers located in different parts of the network.

Distribution concepts. Information distribution is controlled by filtering/re-routing the traffic at the application level. The example given in Section 5.2 shows how a booster can forward information only to parties that have a direct interest in it.

Scalability. Delegating part of the application logic to the boosters enables the information to be treated in a parallel fashion across the network addressing the scalability issue.

Security. As for the security issues, booster boxes only forward data to those recipients that actually should receive it, this is in contrast to existing games, such as *Quake*, that send all information to all participants, trusting the game logic running on the players host to ensure that only appropriate information is displayed. Such games are susceptible to cheating, as "enhanced" versions of the game, e.g. permitting players to see through walls, get written and distributed. Booster boxes run under the control of ISPs, and therefore their software cannot be tampered with.

A concept similar to booster boxes was presented in 1995 by Funkhouser [8]. He proposes an approach that consists in placing "Message Servers" in the network. Each one these entities is in charge of a number of clients and manages message communications on their behalf. In addition Message Servers can perform specific processing on the information. This approach reduces significantly the server load. A major advantage of booster boxes over Message Servers is their network awareness. Booster boxes have the capability of building an overlay network. This BON provides functions such as service discovery or QoS-aware forwarding. Moreover, booster boxes can benefit from NP technology which is an enabler for deep packet-processing at line speed.

Numerous attempts have been made to make networks more programmable [19, 13, 2, 20]. Two types of approaches can be distinguished: those that allow the data path to be programmed [19, 2] and those that restrict programmability to the control path [13, 20]. The former are often called "active networks", the latter "programmable networks". The main criticisms of active/programmable networks are (a) security concerns, (b) lack of convincing use cases, and, for active networks, (c) efficiency concerns. The work described in this paper to boost application performance by taking advantage of application knowledge is influenced by work in this field. Let us briefly answer the three cases just raised.

- Arbitrary third parties are not allowed to run boosters so although booster boxes need to be protected against intruders, security concerns are no greater than those for other pieces of network equipment.
- The active networks literature tends to stress enhanced network control and management functions. Although clearly useful for testing new ideas quickly, successful enhancement can be incorporated into the "standard" network software, thereby removing the need for active networks outside of research laboratories. Booster boxes offer support for specific classes of applications. Such applications are expected to evolve more quickly than control and management functions, are not feasible without some degree of programmability. Moreover, they can generate revenue.
- Whereas some degree of data-path programmability is required to create the virtual overlay, the slow and complex control functions, e.g. link metric calculation, are performed in the booster box control point out-of-band, which then instruments the data path. The small amount of logic required in the data path, e.g. creating packet encapsulations, can be performed efficiently by NPs.

7. CONCLUSION AND OUTLOOK

This paper describes why we think that larger and more complex distributed games than those currently available will require network support to make them feasible. We have presented early results in building a network-aware general-purpose computation platform, called booster box, that provides such a support through application-specific pieces of code called boosters. Booster boxes are co-located with routers and are based on programmable network processors in order to achieve adequate performance.

We presented several scenarios in which booster boxes can be used to provide a scalable solution. While the "large

interactive game-show scenario" has been implemented and tested in a prototype environment, the other scenarios are currently being studied and developed.

8. REFERENCES

[1] EverQuest. http://www.everquest.com, 2002.

[2] D. Alexander, M. Shaw, S. Nettles, and J. Smith. Active Bridging. In *ACM SIGCOMM 1997, Cannes, France*, September 1997.

[3] AMI-C. *Use Cases Release 1 SPEC 1003*. Automobile Multimedia Interface Consortium, 2001. http://www.ami-c.org.

[4] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles*, Banff, Canada, 2001.

[5] ATM Forum. Private Network-Network Interface Specification - Version 1.0 (P-NNI 1.0). *The ATM Forum: Approved Technical Specification*, March 1996. af-pnni-0055.000.

[6] P. Bettner and M. Terrano. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. In *The 2001 Game Developer Conference Proceedings*, San Jose, CA, Mar. 2001.

[7] J. Challenger, P. Dantzig, and A. Iyengar. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. In *Proceedings of ACM/IEEE Supercomputing '98 (SC98)*, Orlando, Florida, Nov. 1998.

[8] T. Funkhouser. Network Services for Multi-User Virutal Environments. In *IEEE Network Realities*, Boston, MA, Oct. 1995.

[9] The Gnutella Protocol Specification v0.4. http://www.clip2.com/GnutellaProtocol04.pdf. Document Revision 1.2.

[10] V. Jacobson. How to Infer the Characteristics of Internet Paths. Presentation to Mathematical Sciences Research Institute, Apr. 1997. ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf.

[11] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. In *ACM SIGCOMM 2000, Stockholm, Sweden*, pages 175–187, 2000.

[12] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *ACM SIGCOMM 2000, Stockholm, Sweden*, pages 283–294, 2000.

[13] A. Lazar. Programming Telcommunication Networks. *IEEE Networks*, 11(5):8–18, September/October 1997.

[14] IBM PowerNP NP4GS3. http://www-3.ibm.com/chips/products/wired/products/np4gs3.html.

[15] G. Regnier. CSP: A System-Level Architecture for Scalable Communication Services. *Intel Technology Journal*, 5(2), 2^{nd} Quarter 2001. http://developer.intel.com/technology/itj/q22001.htm.

[16] N. Shah. Understanding Network Processors. Technical report, University of California at Berkeley, 2001. http://www-cad.eecs.berkeley.edu/~niraj/web/research/network_processors.htm.

[17] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of Networking in Multiplayer Computer Games. In L. W. Sing, W. H. Man, and W. Wai, editors, *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, pages 74–81, Hong Kong SAR, China, Nov. 2001.

[18] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox Communication Architecture and Framework. Internet Draft, Internet Engineering Task Force, Dec. 2001.

[19] D. Tennenhouse and D. Wetheral. Towards an Active Network Architecture. *ACM Computer Communications Review*, 26(2):5–18, Apr. 1996.

[20] J. van der Merwe, S. Rooney, I. Leslie, and S. Crosby. The Tempest - A Practical Framework for Network Programmability. *IEEE Networks*, pages 2–10, May/June 1998.

[21] Wanadoo. Résultats annuels 2001. Presentation to Shareholders, Paris 20th February 2002, French in text, Feb. 2002.

[22] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing . In *Journal of Future Generation Computing Systems*, 1998.