IJCOT -Special Issue– The Malla Reddy National Conference on Information System and Knowledge Engineering (MRNC-ISKE 2013) - July 2013

Interactive Network Applications by LEQ Service

Ch.Rachel Tabitha*1, S.Md.Ibrahim*2

M.Tech, Dept of CSE, SKDEC, Gooty, D.t: Anantapuram, A.P, India

M.Tech, Assistant Professor, Dept of CSE, SKDEC, Gooty, D.t: Anantapuram, A.P, India

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. 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 them1. 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.

II.RELATED WORK:

An excellent overview of the problems arising with the development of networked multi-player computer games can be found in. Smed et al. distinguish four major areas affected 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.

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 Bunkhouses. He proposes an approach that consists in placing "Message Servers" in the network. Each one these entities are 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. Two types of approaches can be distinguished: those that allow the data path to be programmed and those that restrict programmability to the control path The former are often called "active networks", the latter "programmable networks

2.1 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.

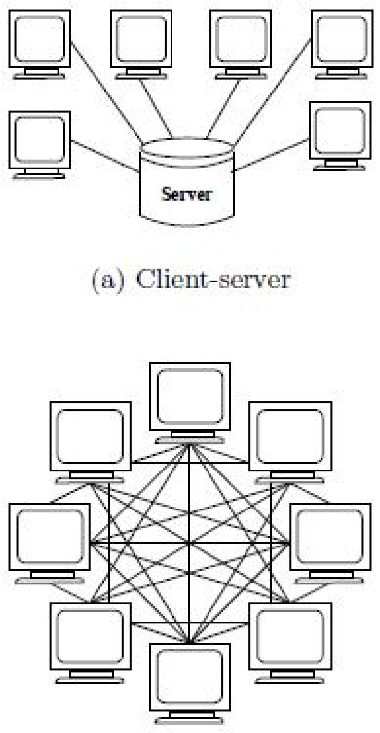

Figure 1: Event-distribution model

2.2 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. Rangier bench-marked an Apache HTTP server running on a Linux PC with a Pentium III 800 MHz processor

2Game events typically describe changes of state, such as a figure moving in a virtual environment. 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 105 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. 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.

2.3 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.

III. 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.

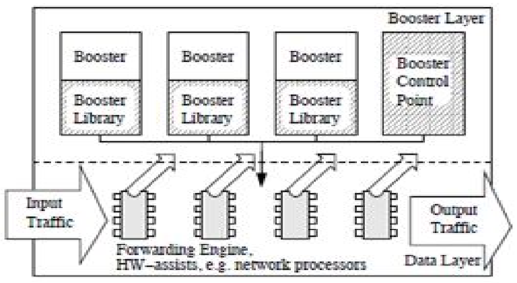

Fig 2: Data Layer Overview

3.1 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.

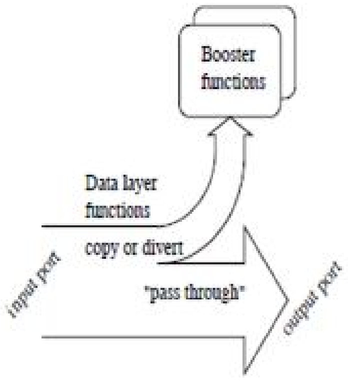

Fig 3: Traffic Forwarding Overview

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 ip tables 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 coprocessors, 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 areshielded 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.

IV. USING THE BOOSTER BOX

In this section we motivate the general architecture 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.

4.1 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/na, 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. Figure 7 shows the general architecture of the IBM NP4GS3. The NP basically consists of line interfaces, a set of picoprocessors (EPC), and an embedded PowerPC processor. Packets are processed at line-speed by the pico-processors3, which are programmed with a low-level language called picocode. 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

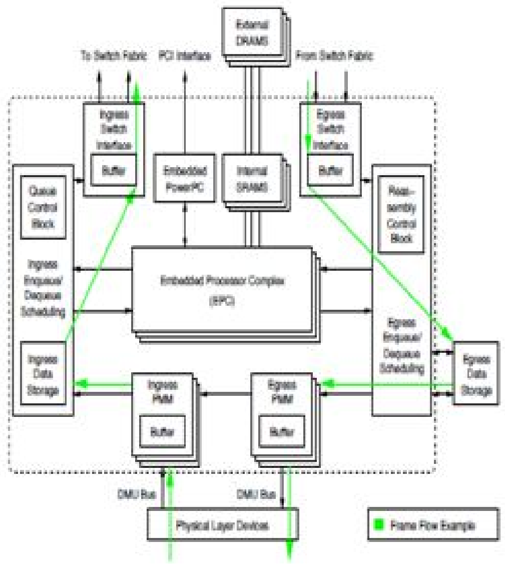

Fig4: General Architecture of IBM NP4GS3

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.

V. CONCLUSION

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.

VI. REFERENCES

[1] EverQuest. http://www.everquest.com, 2012.

[2] D. Alexander, M. Shaw, S. Nettles, and J. Smith. Active Bridging. In ACM SIGCOMM 20087, Cannes, France, September 2008.

[3] AMI-C. Use Cases Release 1 SPEC 1003. Automobile Multimedia Interface Consortium, 2011. 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.

Ch.Rachel Tabitha M.Tech,
SRI KRISHNA DEVARAYA ENGINEERING COLLEGE,
JNTU, Ananthapuram.

S.Md.Ibrahim M.Tech,
Assistant Professor, Department of CSE, SRI KRISHNA DEVARAYA ENGINEERING COLLEGE,
Gooty, Ananthapuram (Dt).