

Tamanoir-IBP: Adding Storage to Active Networks

Alessandro Bassi¹, Jean-Patrick Gelas², and Laurent Lefèvre²

¹ LoCI laboratory

203 Claxton Building, 1122 Volunteer Blvd., University of Tennessee, 37996-3450 Knoxville, TN, USA

abassi@cs.utk.edu

² RESAM Lab. /INRIA

Ecole Normale Supérieure de Lyon - 46, allée d'Italie 69364 LYON Cedex 07 - France

Jean-Patrick.Gelas@ens-lyon.fr Laurent.Lefevre@inria.fr

Abstract. Active networks are a promising way to develop new services for data transport. But active routers could also store “on the fly” streams of data to propose high level services : web cache, reliable multicast... In this paper, we describe the merging of Active networking with Distributed storage solutions. We focus our approach on two dedicated frameworks : the IBP suite¹ and the Tamanoir execution environment². We describe the overall architecture of the Active Logistical Storage suite and present first experimental results.

Keywords : Active networks, logistical storage, Tamanoir, IBP

1 Introduction

Active Networks[TW96], allowing users or applications to inject customized programs into the nodes of the network, are considered a promising framework to enhance the common idea of what a network can do. The creation of new services, such as on-the-fly compression

¹ This work is supported by the National Science Foundation Next Generation Software

² This work is supported by the RNTL Etoile.

or encryption, and what is more important a new way to think about development and deployment of customized modules to perform computation within the network can lead to massive improvements to network functionality.

The Tamanoir project, based on active networks where a new generation of network equipments (such as routers, proxies, and gateways) apply services on the fly on data packets, deals with, in a successful way, some of the historical problems that this kind of approach has to face, such as security and high performance, implementing an efficient multi-streams active transport and dynamic services deployment in the net. Tamanoir services may operate either at a lower level (packets marking, selective drops ...) or at a higher one (QoS, cryptography, compression on the fly ...).

While the above results lie in the active networking field, and can be seen as an optimization of existing ideas, Tamanoir has an original and unexplored approach with regards to on-the-fly storage in the network, as it takes advantage of the Internet Backplane Protocol (IBP) project results concerning network storage management. This project aims to expose network storage resources in an Internet-style way, and if we think about the incredible success of the Internet, which is based on the sharing of resources such as wires and routers, it is absolutely natural to think that sharing more hardware leads, as it does in experimental results, to very positive results.

This paper is organized as follows: section 2 describes the involved frameworks (Tamanoir and the IBP suite). Section 3 focuses on the implementation of a network storage service in an active router; section 4 describes new network services, while section 5 presents our first

performance results. We briefly describe related projects in section 6. In the last section, we present our future works.

2 Background: Tamanoir and IBP frameworks description

This section aims to describe the two projects. The TAMANOIR active networking execution environment, and the Logistical Networking Infrastructure, in which the Internet Backplane Protocol is the basic building block.

2.1 Tamanoir : High Performance Active Networking

The integration of new and standard technologies into the shared network infrastructure has become a challenging task, and the growing interest in the active networking field[TW96] might be seen as a natural consequence. In “active” networking vision, routers or any network equipments (like gateway or proxy) within the network can perform computations on user data in transit, and end users can modify the behavior of the network by supplying programs, called *services*, that perform these computations. This kind of routers are called *active nodes* (or *active routers*), and show a greater flexibility towards the deployment of new functionalities, more adapted to the architecture, the users and the service providers’ requirements.

The Tamanoir architecture

The Tamanoir[GL00,GL01] architecture design does not interfere with the core network,

mainly to guarantee higher performance results, and it's deployed only on the network periphery (Fig. 1).

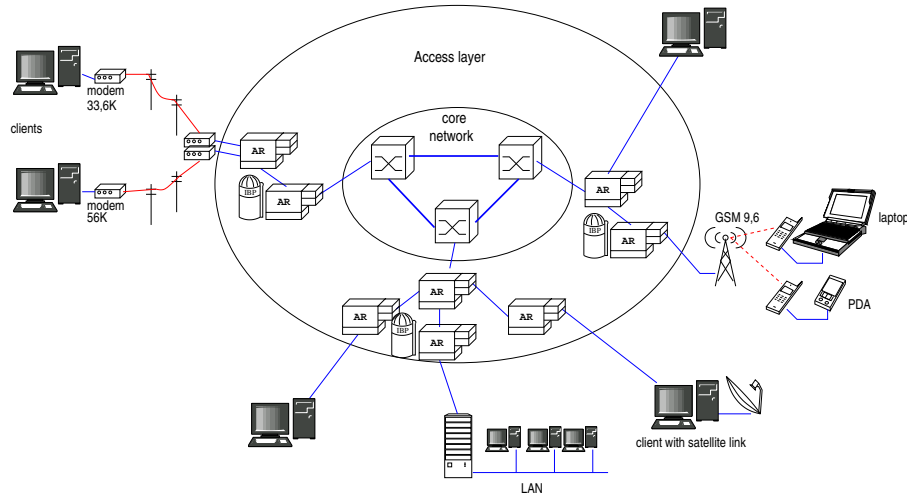


Fig. 1. Tamanoir active network topology. Active Routers are edge routers. IBP depots are distributed close to the Tamanoir active nodes

The injection of new functionalities, called services, is independent of data stream: services are deployed on demand when streams reach an active node (see Figure 2) which is not able to perform the expected data process.

Tamanoir Active Nodes (TAN) provide persistent active routers which are able to handle different applications and various data stream (such as audio/video) at the same time. The two main transport protocol (TCP and UDP) are supported by the TAN for carrying data. We use the ANEP (Active Network Encapsulated Protocol)[ABG⁺97] format to send data over active networks.

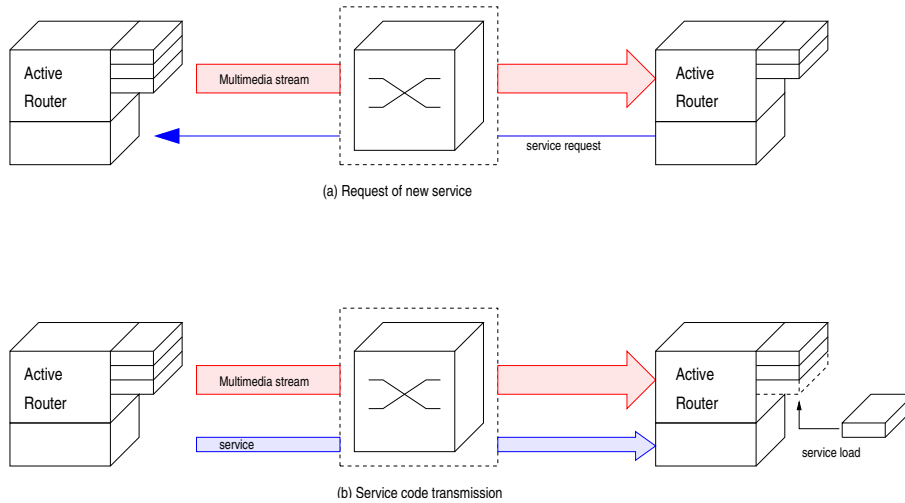


Fig. 2. Service deployment between two Active Routers. When a stream reach an active routers who doesn't hold the required service, it sends a request to the last active router crossed by the stream. Then, the service is sent, uploaded, installed in memory and ready to process the stream. A stream can cross equally a classical router, without any processing actions.

For the implementation process we choose to use a common and portable language to let active networks users be able to define and write their own services. For this reason the TAMANOIR execution environment has been written entirely in JAVA [jav], because this language provides great flexibility typical of an Object-Oriented language and is shipped with standard library.

Unfortunately, the execution environment provided by the JVM (*Java Virtual Machine*) gives a very high level of abstraction, through which applications have some difficulties to reach the guts of a system. In the context of high performance this abstraction can raise a huge number of optimizations problems. Luckily, recent JVM releases ($\geq 1.3.x$) provided by company like *SUN*[SUN], *IBM*[IBM] or by project like *Blackdown*[Bla] give excellent performance for the mainstream hardware architecture (i.e., x86). This is mainly due to

the improvements in Just-In-Time (JIT) compilation techniques. It is important to notice that the *GNU Compiler for the Java Programming Language*, called GCJ[GCJ], can be used to run a TAN in native mode.

As shown in Figure 3, a TAN is divided in two parts; the main part, called *TAMANOIRd*, routes packets towards the adapted service in function of a hash key contained in the packet's header. New services are plugged in it dynamically. After being processed, the packet is forwarded to the next TAN(s) in function of an internal routing table of deployed TANs. The second part, called Active Node Manager (ANM), is dedicated (1) to send services request to another TAN if the one doesn't have the appropriate service installed for a particular stream and (2) to update its routing table.

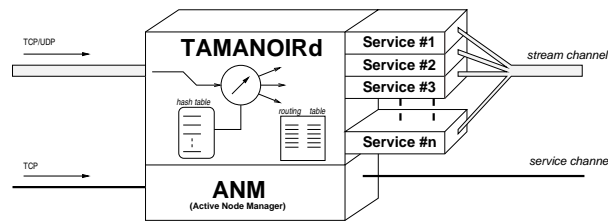


Fig. 3. A Tamanoir Active Node (TAN)

2.2 The Logistical Networking Infrastructure and IBP

IBP[PBE⁺99] is a middleware built to share distributed storage resources of any size in any network size. What makes IBP unique is the way of *exposing* those resources: any application can allocate a predictable amount of space for a predictable amount of time on a given server. This key aspect of the IBP storage model, the capacity of allocating

space on a shared network resource, can be seen as doing a C-like malloc on an Internet resource, with some outstanding differences, such, for instance, time-limitation to prevent Denial-of-Use attacks.

From a networking point of view, IBP design matches in many ways the design of the Internet Protocol (IP); as IP is a more abstract service based on link-layer datagram delivery, IBP is a more abstract service based on blocks of data, managed as “byte arrays”.

As in the networking field the IP protocol alone does not guarantee many highly desirable characteristics, and needs to be complemented by a transport protocol such as TCP, in a similar way many application who might need stronger allocation properties, such as reliability and arbitrary, might find IBP semantic too weak.

For this reason, the *exNode*, or *external Node*, following the example of the Unix inode, aggregates sets of storage allocations on the Internet (rather than aggregating blocks on a single disk volume). As its definition is very generic any kind of storage allocation can be suited; but the exposed nature of IBP allocations makes IBP byte arrays perfectly fit for such aggregations.

The IBP software, developed in C (for the server side) and C and java (for the client library), has been successfully tested for different OSes and hardware architectures (Linux on i686 and ia64, Solaris, Windows 2000,AIX, DEC alpha, OS X).

The exNode library (developed in C and java) allows to serialize its contents using XML, to open the exNode use to XML-based tools and inter-operability with XML systems and protocols.

3 Implementation : Design of IBP Services for Tamanoir

A Tamanoir Active Node (TAN) is an autonomous process running on a node (PC) waiting for processing new active streams. An IBP depot is an autonomous process which can run on the same or another local node. As shown in figure 4, Tamanoir and its services run in the application layer (the higher rectangle) and access directly the IBP layer.

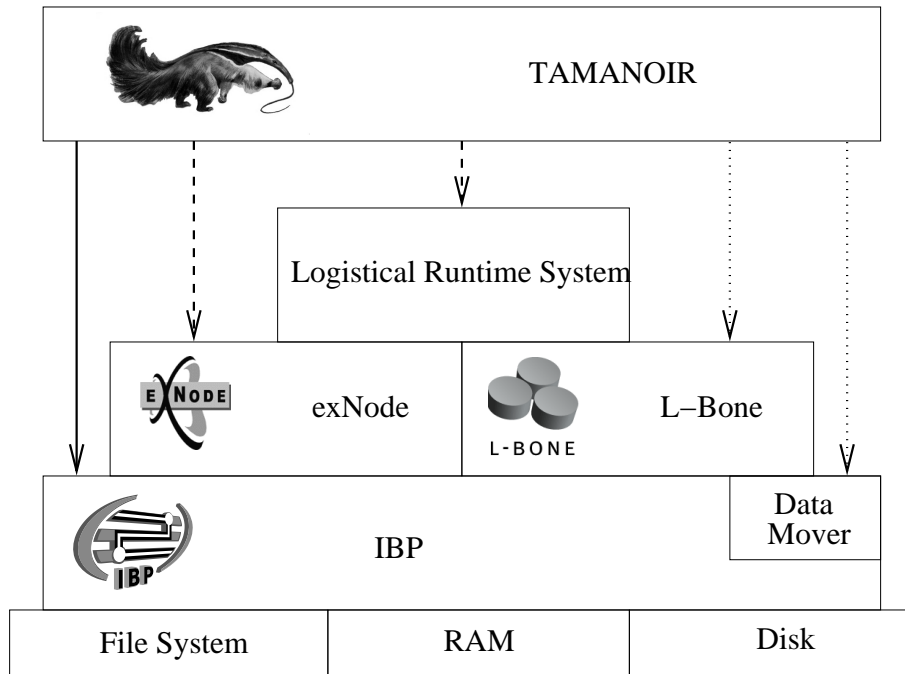


Fig. 4. Tamanoir - IBP architecture

A Tamanoir service communicates with an IBP depot through a socket over a reliable transport protocol (TCP/IP). It is worth noticing that it is not mandatory for an IBP depot to run on the same node, because communications between depot and active service are done through the intermediary of sockets; but, of course, this architecture might show a better performance. The original Java client for IBP was re-used to implement our code,

as a Tamanoir service is considered as a client from an IBP depot point of view. We have to instantiate IBP client classes in each service requiring to communicate with an IBP depot. These classes provide constructors and methods to create *Capabilities* (pointers to IBP allocations) on any IBP depot, with whom the Tamanoir service can write, read and manage data remotely on any IBP depot.

Figure 5 presents a very basic service for the Tamanoir Active Node. This service is designed to cache data in a IBP depot and forward them immediately. Each service is inherited from a generic Service which is a *Thread*. When this service is instantiated, the constructor `Ibp_testS()` (in our example) is called. Its goal is to allocate a storage area in the chosen IBP depot, and returns the pointers (or capabilities) to this area. When this is done, the main loop `run()` copy the ANEP payload to the IBP depot and forward the entire ANEP packet to the receiver.

4 Network services

In this section, we illustrate the current services and discuss about the future ones, which can be used by applications willing to take advantage of the mixture of the IBP storage capabilities and TAMANOIR dynamic behavior alteration capabilities.

4.1 Data caching during service deployment

As the TAMANOIR project was planned essentially as a high performance active networking environment for the transmission of multimedia streams, using some unreliable protocol like UDP for applications like VoD (Video on Demand), the packet loss was toler-

```

class Ibp_testS extends Service
{
    static final String IBP_DEPOT_NAME = "ibp01.ens-lyon.fr";
    static final int IBP_DEPOT_PORT = 6716;
    static final int CAP_SIZE = 1000 ; // KBytes
    IBPDepot dpt ;
    CapAttribute attr ;
    IBPCaps caps ;
    WriteCap wc ;
    ReadCap rc ;

    Ibp_testS()
    {
        try{
            dpt = new IBPDepot( IBP_DEPOT_NAME ,IBP_DEPOT_PORT );
            attr = new CapAttribute(0,IBPProtocol.IBP_STABLE,
                IBPProtocol.IBP_BYTEARRAY);
            caps = new IBPCaps(dpt,CAP_SIZE*1024,attr,0,0);
            wc = caps.getWriteCap();
            rc = caps.getReadCap();
        } catch (UnknownHostException e) {
        } catch (IBPException e) {...}
        } catch (IOException e) {...}
    }
    public void run()
    {
        long ret1 = 0;
        while(!FINISHED) {
            try {
                // 1/ cache payload in the local ibp depot
                ret1 = wc.write( getANEPkt().payload,
                    getANEPkt().payload.length );
                System.err.println("Ibp_testS: write in " + IBP_DEPOT_NAME +
                    " depot : " + ret1 + " bytes");
                // 2/ send cached data immediately
                forward();
            } catch (IOException e) {
                System.err.println("Ibp_testS 1: " + e);
            } catch (IBPException e) {
                System.err.println("Ibp_testS 2: " + e);
            }
            // 3/ wait for the next ANEP packet
            rcv();
        } // while(!FINISHED)
    } // run()
    public void process() {}
} // class Ibp_testS

```

Fig. 5. IBP service in TAN

ated. On the other hand, what was not acceptable was losing packets because the service that should have taken care of the data was not present. Therefore, when a Tamanoir Active Node (TAN) received a stream without holding the service needed, the first packets were simply discarded until the service was loaded and installed in the TAN memory. To correct this behaviour, we run an IBP depot on the local node (where the TAN is running) in order to cache data (see Fig.6).

This first new service, called *IBPService* uses a *IBP_store* operation to redirect the data stream beginning towards the IBP depot. The IBP service checks as well the presence of

the required service each time that a new packet arrive, and if so, a *IBP_load* operation is done to redirect all the data cached in the IBP depot towards the service able to process, route and forward efficiently these data. The only difference between the IBPService and any other service lies in the load-time, which is done at boot time for the IBPService, in order to be able to cache data immediately.

This service is also useful for a reliable transport protocol such as TCP. As we said above, until the service was loaded and ready to operate, all the packets were dropped. In case of a reliable connection, as we don't have the "right" to loose data, the TAN should send a message to the server to inform it about the loss of packets and ask him to retransmit. As above, with an IBP depot this is not necessary because data are cached in the local depot until the TAN is able to process and transmit them efficiently. With a FIFO spirit, cached data are sent first, and if the capability gets empty, the remaining data are forwarded to the service without passing through the local IBP depot.

4.2 Reliable Multicast

There are three benefits of performing storage in the network for a reliable multicast application. First, we can look at the TAN with its depot as a kind of mirror for data distribution, to download them from the geographically (or network) closest point to the consumers. Next, clients can consume data with their own processing speed capabilities without disturbing the server where data come from, and finally, a TAN can retransmit lost data without uselessly overloading the network between the server and the TAN.

The above considerations led us to design a second service, called *XcastIbpService* and it is targeted for multicast applications. Figure 6 shows a view of multicast topology with on the left a data server (e.g push server, ASP server) and on the right heterogeneous clients in terms of connections speed and processing capabilities. Because clients don't ask the same data at the same time and don't consume them at the same speed we put an IBP depot in (or close to) a TAN to cache data. For this application, a TAN can be seen has a multicast node with caching capabilities. Server have just to send only once data towards the closest active node to the clients. Then, clients can consume data at their own speed.

For the first experiments, we implemented a very simple multicast algorithm, but we are currently implementing a more complex algorithm of reliable multicast called DyRAM (Dynamic Replier Active Reliable Multicast)[MP01].

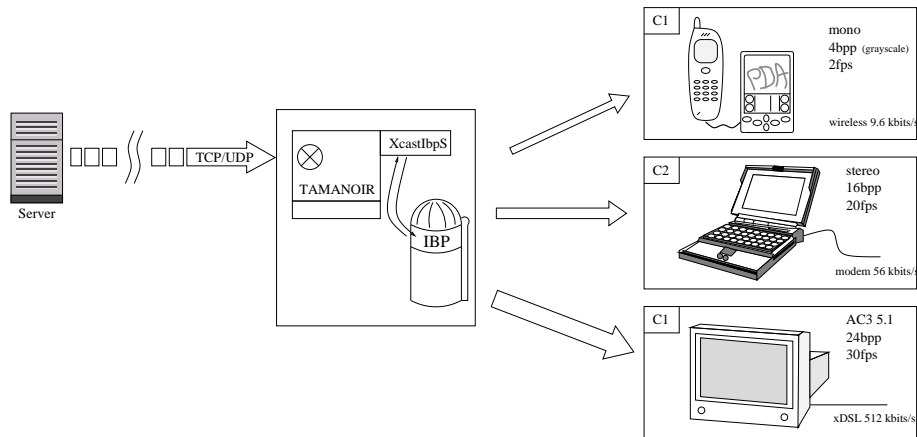


Fig. 6. Reliable multicast with Tamanoir and IBP depot

4.3 IBP-mail done with Tamanoir

IBP-mail [EPBW99] allows a user to send an attachment of whatever size without overloading the mail server by making use of the network storage available through a set of IBP storage depots. The file is stored in a IBP depot for the duration specified. The receiver can download the file at its convenience, using a tool developed for this purpose.

For this application we would like to leverage the processing capability of a TAN for managing the movement from an IBP depot close to the sender to an IBP depot close to the receiver in order to take advantage of proximity (i.e high throughput and low latency) of the attachments, which are usually big.

This kind of application involves the use of processing capabilities in network to route attachment towards the nearest IBP depot to the receiver. The notion of proximity in computer networks is still an active open problem [Swa99] and can be measured with different metrics (round-trip time, latency, throughput, space, hops numbers...)

Since the need to move very large digital objects around within the science, engineering, financial and even graphic multimedia community is getting stringent, we expect that the capacity to manage extremely large attachments will be fundamental for the e-community.

5 Raw performance results

In this section we measure the overhead introduced by the caching action in an IBP depot close to the Tamanoir Active Node. For these results we ran an IBP depot on the same node than the TAN, with the *XcastIbpService* (described in section 4.2).

Our testbed, disconnected from the production network, is set up with one active Tamanoir node which is a PC (Dual-Pentium III, 1 Ghz, 256MB RAM) shipped with several *Ethernet 100Mbits* network interface cards and a standard IDE hard-drive. We link to this PC different other PC who have the role to either feed the network or receive data. These PCs run under GNU/Linux with a 2.4.16 kernel (distribution *Debian 2.2r3*).

When an ANEP packet reaches the TAN, its payload is extracted and sent towards the required service. At the beginning, the service copies the payload in the local IBP depot. For the measures we have set up only one client which consume data as fast as possible. Next, the payload is read in the cache and forwarded to the client immediately. For each packet, we measure the time to cross a TAN (latency) with and without caching action. A chronometer is located as low as possible in the kernel. We use *Netfilter* [Rus00] which is a package for filtering in the Linux kernel. NetFilter allows to introduce code just behind the network interface card. When a packet reach the TAN, our NetFilter module analyses the packet, if it is an ANEP packet a chronometer is started. When the same ANEP packet leave the TAN, the chronometer is stopped.

In figure 7 we present the latency of a TAN by an ANEP packet. In the first case (lower curve) packets are immediately forwarded and not cached in the IBP depot, in the second case packets are cached and next forwarded.

As shown, caching doesn't introduce a very important overhead. For any packet size, overhead remains constant. For small packets size, performances are weak due too the policy of transmission of Linux kernel TCP implementation which tries to aggregate small packets before being transmitted. So, small packets are released just after a timeout.

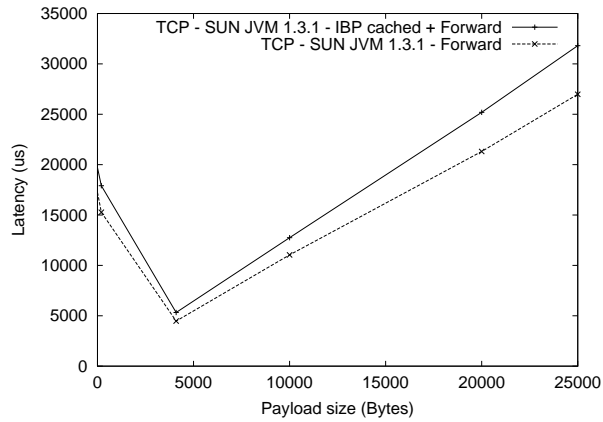


Fig. 7. Overhead introduced by caching data in a local IBP depot

6 Related works

Despite intensive search, we do not find any related projects directly merging distributed logistical storage with active networks technology.

IBP occupies an architectural niche similar to that of network file systems such as AFS[MMM⁺86] and Network Attached Storage appliances , but its model of storage is more primitive, making it similar in some ways to Storage Area Networking (SAN) technologies developed for local networks. In the Grid community, projects such as GASS [JIC⁺99] and the SDSC Storage Resource Broker [CRAM98] are file system overlays that implement a uniform file access interface and also impose uniform directory, authentication and access control frameworks on their users.

Active networks projects using storage facilities are not intensively addressed. Most related projects are Active Disks [Rie99] or active-network-storage [TTI⁺00] which explore the increasing of intelligence in Network Attached Storage.

7 Conclusion and future works

By merging two environments like IBP and Tamanoir, we allow users and network operators to provide new kind of high level services.

The benefits for the frameworks are double. Tamanoir can use storage distributed facilities (IBP depots) for service deployment and temporary storage of data streams. It is now easy to transform an active router in an active proxy with on the fly storage capabilities. The latency added by a dynamic storage is insignificant. IBP benefits from Tamanoir dynamic service deployment infrastructure.

As adding storage in the network could also mean to add disks in the routers, which is an unusual approach for networks operators (low MTBF, failure...), a RAM-based version of IBP is currently under design by LoCI Laboratory.

Our next step will consist in the design of high level network services dedicated to Grid computing middleware requirements.³

References

- [ABG⁺97] Scott D. Alexander, Bob Braden, Carl A. Gunter, Alden W. Jackson, Angelos D. Keromytis, Gary J. Minden, and David Wetherall. Active network encapsulation protocol (anep). RFC Draft, Category : Experimental, <http://www.cis.upenn.edu/switchware/ANEP/>, July 1997.
- [Bla] Blackdown. Blackdown java development kit. <http://www.blackdown.org/>.

³ More information on both frameworks can be found on the following sites :

- Tamanoir : http://www.ens-lyon.fr/LIP/RESAM/index_activenetwork.html
- IBP : <http://loci.cs.utk.edu/ibp>

- [CRAM98] C.Baru, R.Moore, A.Rajasekar, and M.Wan. The SDSC Storage Ressource Broker. In *CASCON'98*, Toronto, Canada, 1998.
- [EPBW99] Wael Elwasif, James Plank, Micah Beck, and Rich Wolski. Ibp-mail: Controlled delivery of large mail files. In *NetStore'99: Network Storage Symposium*, Seattle, WA, October 1999.
- [GCJ] GCJ. The GNU Compiler for the Java Programming Language. <http://sourceware.cygnus.com/java/>.
- [GL00] Jean-Patrick Gelas and Laurent Lefèvre. Tamanoir: A high performance active network framework. In C. S. Raghavendra S. Hariri, C. A. Lee, editor, *Active Middleware Services, Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 105–114, Pittsburgh, Pennsylvania, USA, August 2000. Kluwer Academic Publishers. ISBN 0-7923-7973-X.
- [GL01] Jean-Patrick Gelas and Laurent Lefèvre. Mixing high performance and portability for the design of active network framework with java. In *3rd International Workshop on Java for Parallel and Distributed Computing, International Parallel and Distributed Processing Symposium (IPDPS 2001)*, San Fransisco, USA, April 2001.
- [IBM] IBM. Ibm java development kit. <http://www-106.ibm.com/developerworks/java/jdk/linux130/>.
- [jav] Java programming language. <http://java.sun.com/>.
- [JIC⁺99] J.Bester, I.Foster, C.Kesselman, J.Tedesco, and S.Tuecke. Gass: A data movement and access service for wide area computing systems. In *Sixth Workshop on I/O in Parallel and Distributed Systems*, may 1999.
- [MMM⁺86] J.H. Morris, M.Satyanarayan, M.H.Conner, J.H. Howard, D.S.H. Rosenthal, and F.D. Smith. Andrew: A distributed personal computing environment. *Communication of the ACM*, 29(3):184–201, 1986.
- [MP01] M. Maimour and C. Pham. A throughput analysis of reliable multicast protocols in an active networking environment. In *Proceedings of the Sixth IEEE Symposium on Computers and Communications (ISCC 2001)*, pages 151–158, July 2001.
- [PBE⁺99] James Plank, Micah Beck, Wael Elwasif, Terence Moore, Martin Swany, and Rich Wolski. The internet backplane protocol: Storage in the network. In *Dept. of CS, University of Tennessee*, 1999.
- [Rie99] Erik Riedel. Active disks - remote execution for network-attached storage. Technical Report CMU-CS-99-177, Electrical and Computer Engineering Carnegie Mellon University, Pittsburgh, PA 15213, Nov. 1999.
- [Rus00] Rusty Russell. Linux Filter Hacking HOWTO. netfilter description and usage, july 2000.

- [SUN] SUN. Sun java development kit. <http://java.sun.com/>.
- [Swa99] M. Swamy. Network proximity resolution with sonar and sonardns. Technical Report CS-99-429, University of Tennessee, January 1999.
- [TTI⁺00] Aki Tomita, Yoshifumi Takamoto, Shigekazu Inohara, Hiroaki Odawara, Frederico Maciel, Mamoru Sugi, and Naoki Watanabe. A scalable, cost-effective, and flexible disk system using high-performance embedded-processors. In *IEEE International Conference on Parallel Processing*, 2000.
- [TW96] David Tennenhouse and David Wetherall. Towards an active network architecture. *Computer Communications Review*, 26(2):5–18, April 1996.