Appeared in: Proceedings of the 20th International Conference on Distributed Computing Systems,
Taipei, April 2000.

HYDRANET-FT: Network Support for Dependable Services

Gurudatt Shenoy Suresh K. Satapati Riccardo Bettati
Department of Computer Science
Texas A&M University

Abstract

With the Internet increasingly being used as access medium for a variety of critical services, there is a growing need to provide fault tolerant services over internetworks, in a completely client-transparent fashion. We present HYDRANET-FT*, an infrastructure to dynamically replicate services across an internetwork and have the replicas provide a single fault tolerant service access point to clients.* HYDRANET-FT *uses the TCP communication protocol with a few modifications on the server side to allow one-to-many message delivery from a client to service replicas and many-to-one message delivery from the replicas to the client. A communication channel between the replicas provides atomicity and message ordering. A low-latency failure estimator is used to detect failures of servers in the system and initiate fail-over mechanisms. An implementation and measurements on a local testbed show that the overhead of our scheme is reasonably small.*

1. Introduction

The Internet is increasingly being used as access medium for a variety of critical services, for which very high requirements are set on availability and reliability. Examples range from e-commerce sites, to per-pay media servers, to information servers in distributed mission critical systems, such as air-traffic control systems. During live Web broadcasts of important events, for example, the video service serving potentially many thousands of clients with live action must guarantee uninterrupted broadcast. Similarly, service interruptions for an on-line brokerage firm may have very serious effects. To satisfy the fault-tolerance requirements in such systems, an infrastructure must be in place that is able to effectively handle failures in the network and at servers with little to no effects on clients. As service disruption can be caused by temporary overload as well, this infrastructure must also be able to diffuse extreme load conditions in the network and on the servers.

The initial emphasis of services over internetworks was merely to provide high availability (ensuring that a service was available to clients regardless of the load). To this end, a number of schemes were adopted, including DNS caching, client and proxy caching, and IP-level service replication. In case of failure of the service mid-way through a client interaction however, it was left to the client (typically the user) to seek the service again later or from a different source. Of late, the issue of providing fault-tolerant services over internetworks in a client-transparent fashion is rapidly becoming an important area, driven by a need for such services on the Web.

In an attempt to meet these needs, a number of approaches to providing fault-tolerant distributed services on the Internet have been proposed. Commercial solutions include clustering [2, 11], hardware support, such as usage of dual-ported disks [3], and others. Such solutions however are susceptible to "site-disaster", For example, the network link to the cluster may fail or simply be temporarily congested, rendering the whole cluster inaccessible and disrupting the service, while nodes in the cluster are functional. Protocols and related toolkits have been proposed that attempt to create a complete environment for supporting fault-tolerant services, e.g., Horus [10], Isis [4], Transis [1], Consul [16], and many others. While these have proved to be quite effective by themselves, they are difficult to deploy for large-scale applications, since they rely on low-level interfaces and may be difficult to transparently integrate with the Internet standard TCP/IP protocol suite. In addition, converting to and from the native Internet protocols adds overhead, thereby affecting performance.

An infrastructure is needed that provides all the essentials to deploy fault-tolerant services across an internetwork. For this, it must provide a number of capabilities. First, it must provide *service replication* across geographically distributed server hosts, which in turn may be served by different network providers. This is needed to increase resiliency to failures or congestions in servers or portions of the network. Second, it needs support for *atomic multicasting* to ensure that all servers agree on the set of operations to perform on behalf of the clients. Third, *message ordering* is necessary to ensure that all servers agree

on the order of events to be dispatched to the server programs. Next, a *low-latency failure detection* mechanism is needed that quickly identifies service disruptions. Such a mechanism would then allow to quickly shut-down the misbehaving server, thus effectively providing a fail-stop failure behavior. This is particularly important when servers spuriously become unavailable due to server or network congestion. Ideally, it should be possible to temporarily shut down servers when they cause service disruption due to congestion, and bring them back in when the congestion clear. Finally, and very importantly, the infrastructure must provide full *transparency to clients* in order to enable the deployment of fault-tolerant services with a large existing client population. This is particularly important for Web-based services, where client programs are typically general-purpose Web browsers, which are burdensome to extend to incorporate proprietary protocols for access to replicated servers or for failure recovery. Such client programs should continue to see the expected single service access point, typically in form of a TCP connection to the server.

Additionally, such an infrastructure must be fully application independent, making it a solution for current and next-generation services, and must be easy to incrementally shoehorn into an existing internetworking infrastructure (similar to the MBONE [9]).

In this paper, we propose an infrastructure to dynamically replicate services across an internetwork while having them provide a fault-tolerant single service access point to clients. For service replication, we rely on our previous work on network support for large-scale service scaling within the HYDRANET effort [6]. In this paper we describe how to combine service replication with TCP communication service to provide fault-tolerant services in a fully client-transparent fashion. By this we mean that a fault-tolerant TCP connection is provided between the client and the service, and neither the client application, nor the client TCP stack are aware of service management, server failures, and server recoveries.

In order to test the feasibility of our approach, we implemented HYDRANET-FT, a protocol infrastructure for fault-tolerant service replication. Similarly to HYDRANET, HYDRANET-FT consists of two components: host servers and redirectors. *Host Servers* are hosts that are specially equiped to act as servers for replicated and fault-tolerant services. To achieve transparent service replication, they are able to host IP services that may be known to the outside world under the IP address of another host. [1]

To achieve fault-tolerance for TCP services, the protocol processing on the host server is able to handle TCP communication from clients to a group of replicated servers. To do this, it synchronizes communication between the client and the servers and provides a low-latency failure estimator to determine server unavailability due to failure or congestion.

[1] This is used in HYDRANET to achieve service scaling. Typically, the service running on the server host is then a replica of the service running on the *Origin Host*, for example a mirror of a site of a Web server. It can also be a scaled-down versions of the service (for example an active cache) that runs on the host server as agent of the server on the origin host.

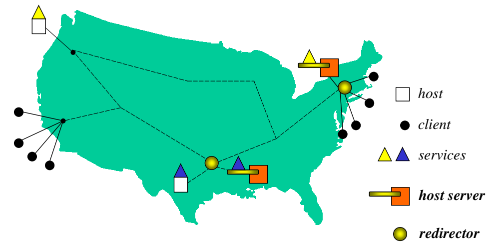

Figure 1. Replicating Services in HydraNet

The location of the host servers is known to the *Redirectors*, specially equipped routers that maintain information about the host servers, replicated services and thos host servers running copies of them. Redirectors detect requests for replicated services, and direct the requests on to the appropriate host servers(s) based on what the requirements of the service are. If the service is simply replicated for scalability, that is, with no fault-tolerance requirements, then the redirector may forward the request to the "nearest" available host server with a replica running. If the service is replicated on several host servers, and is required to provide fault-tolerant service, the redirector sends each of the available host servers a copy of the request.

A replica-management protocol allows to dynamically install services, or service replica, or remove them, depending on the load in the network and on the servers. It also allows to reconfigure the system in case of failure or prolonged congestion of a server.

The scenario in Figure 1 gives a general idea of how the components of HYDRANET-FT work together: The service `www.northwest.com` is accessed by large groups of users from the two ISPs `southwest.net` and `northeast.net`. Without a replication scheme, the distance from the clients in `northeast.net` to the server `www.northwest.com` can cause increased access latencies and network load. In addition, the server itself may be overly loaded. In this example, the ISP `northeast.net` routes its traffic through a redirector, and has access to a host server. The server at `northwest.com` installs a replica for its Web service on the host server of `northeast.net`. We elaborate in [6] on how this approach helps control the load on network and servers, and proactively diffuses hot spots.

The scenario in Figure 1 also shows how a fault-tolerant service is deployed in HYDRANET-FT. The service `audio.south.com` (dark triangle) is replicated on two hosts, of which one is a host server. Both hosts are accessible to all clients through at least one redirector. Whenever

a request to this service reaches the redirector, it gets forwarded to both hosts. Failure detectors on the hosts inform the redirectors about host failures and trigger a reconfiguration: the redirector updates its redirection tables to reflect the unavailability of the failed host. In this way, servers that experience spurious unavailability (this is common during network or server congestion) can be effectively "shut down", allowing for a fail-stop failure behavior.

The rest of the paper is organized as follows. Section 2 discusses work done in related areas. This includes a classification of the approaches commonly adopted to provide fault-tolerant distributed applications. In Section 3 we give a short overview of mechanisms for service replication in HYDRANET. Based on this, we present in Section 4 how we extend service replication to fault-tolerant service replication for TCP services by incorporating support for atomicity, message ordering, and failure detection for TCP-based services into the protocol processing on the host servers. Section 5 provides a number of performance measurements. Section 6 presents the conclusion and the outlook for future work.

2. Related Work

A number of approaches to designing fault tolerant, distributed applications have been explored. In the following we will describe a number of such schemes and group them according to their location within the overall system architecture.

Group Communication Toolkits. A large number of approaches have been proposed to support fault tolerance, load balancing, and service replication through communication subsystems that provide various forms of reliable group communication. Such approaches typically provide process groups and fault-tolerant multicast, and support a number of event ordering protocols. Typically, they also support some form of virtual synchrony. Examples of implementations are Isis [4], Horus [10], Transis [1], Consul [16], and Totem [17]. Unfortunately, service deployment using these toolkits is not easy, as they each rely on their own, low-level, programming interfaces, which typically must be used in both server and client programs, thus making integration of existing client bases difficult.

Primary-Backup Systems. A number of approaches to implement fault-tolerant services using a server replication and a primary-backup paradigm have been proposed [5, 8, 19, 20]. The service is replicated to have one primary and one or more backups. Clients make requests to the service by communicating to the primary. When the primary server fails, one of the backup servers takes over in what is called a *failover*. Similarly the approaches based on group communication described earlier, the client must be well aware of the protocol being used. It also must be aware and maintain the identity of the current server.

Fault-Tolerant Distributed-Object Middleware. It has been argued (e.g., [15]) that the process group abstraction provided by group communication toolkits is too low-level for the development of distributed applications, and should be replaced by an *object-group* abstraction. Examples of systems that integrate distributed object-oriented technology with a fault-tolerant communication subsystem to achieve this are Orbix over Isis and Electra [13], Goofy [7], and Eternal over Totem [18].

3. HydraNet

HYDRANET replicates services by *globally replicating IP addresses*. A service is replicated by installing replicas on one or more *host servers* and have them bind to the same set of TCP or UDP ports as the service on the origin host. *Redirectors* ensure that the replicas on the host servers are accessible under the same IP address as the origin host. When a redirector receives an IP packet destined to a replicated service, it knows the location of the "nearest" replica of the service, which is identified by the pair of IP address and port number. If the destination of the packet does not appear to the redirector as a service with replica, the packet is simply forwarded to its destination. This allows to dynamically, and transparently, install replicas at strategic locations (for example "near" large client populations).

IP-Redirectors. Each redirector maintains a *redirector table*, which lists the transport-level service access points (in our case pairs of IP addresses and port numbers) for which packets must be redirected, and the host server to which the packets must go.

When a redirector receives an IP packet, it checks the destination IP address and port in the header against the entries in the redirector table. If it finds a match, it forwards the packet to the appropriate server host. If there is no match, the packet is simply forwarded to the origin host. A packet is redirected to the appropriate host server by *tunnelling* it using IP-in-IP encapsulation. The destination host server is equipped to detect tunneled packets and to forward them internally to the service.

Host Servers. Replicas of server processes run on host servers, which are specially equipped hosts that act as *servers-of-servers*. A replicated service on the host server runs as a replica of the server program. This server program runs within an environment called a *virtual host*, which is identified by the IP address of its origin host. A new virtual host is created on the host server by the system call

```
int v_host(u_long ip_address);
```

which associates the currently running process with the given IP address. Whenever the process, or any of its descendants, binds a socket to a TCP or UDP port, the socket

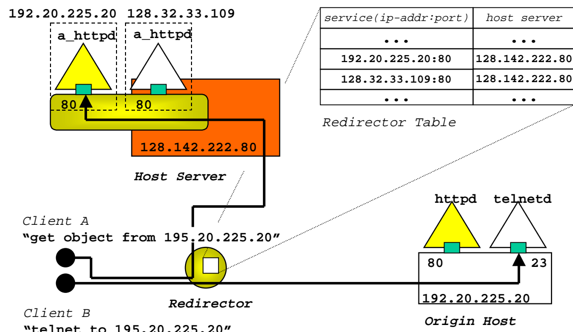

Figure 2. Components of HydraNet

belongs to the virtual host associated with the process. Whenever a socket is created, the kernel checks whether the current process belongs to a virtual host and marks the socket's protocol control block appropriately. The replica management protocol then informs the redirector about the newly created socket on the host server, and the redirector sets up the redirection information appropriately.

When a packet destined to a virtual host is received by the host server, its destination IP address and port number are compared against currently installed virtual hosts and the ports applications are bound to. If a match is found, the data is deposited at the appropriate socket buffer.

Figure 2 shows how the components of HYDRANET interact. We observe from the figure that Host `128.142.222.80` is a host server. The Web service (realized by the `httpd` daemon) on Host `192.20.225.20` is replicated on the host server, where it is realized by the `a_httpd` replica daemon. Whenever the process on the host server binds to a TCP or UDP port, the host server and the redirectors are informed, and the redirector tables updated. HTTP requests (identified by destination port number 80) from Client A are intercepted by the redirector, which happens to be on their route, and which was informed earlier that the nearest Web port for host `192.20.225.20` is located on host server `128.142.222.80`. The requests are routed accordingly. Client B's requests for the telnet service are not rerouted by the redirector, but are forwarded to the origin host; the redirector does not have an entry for the telnet port of host `192.20.225.20`. We note that neither the clients nor the non-participating servers are affected by this scheme.

4. HydraNet-FT

While HYDRANET effectively handles service replication and load balancing, it provides no facility to handle server failures. This has been addressed in HYDRANET-FT, which extends service replication by adding a fault-tolerant TCP communication service. A TCP-based fault-tolerant service is realized by replicating a server program onto one or more hosts and by having all replicas bind to the same TCP port on all the hosts. A fault-tolerant TCP connection has a single service access point (in BSD this is the TCP socket) on the client side, and one or more service access points on the server side, one for each replica. The communication subsystem in HYDRANET-FT provides ordered, atomic, one-to-many message delivery from client(s) to the group of replicas, and many-to-one message delivery from replicas to the client(s). This is accomplished by (i) message replication at the redirectors, (ii) a primary-backup setup on the replica side, with only the primary communicating back to the client while the backups are kept on hot-standby, and (iii) modifications at the TCP machinery to provide an *acknowledgment channel* from backups to the primary for atomicity and message ordering purposes.

We realized HYDRANET-FT as a set of simple modifications to the process management and the TCP/IP protocol stack in the FreeBSD kernel. In this section we describe the most important aspects of the design on the Host Server and the Redirector. We describe the light-weight synchronization mechanisms between servers used to achieve reliable communication. Also, we give a short overview of the replica management protocol to manage replicated servers and their TCP ports.

4.1. Host Servers

In HYDRANET-FT, a replica of the server program for a fault-tolerant service either operates as a primary server or as a backup server. This mode of operation is defined during server program startup by defining the operation mode of the transport service access point of the server, in our case its well-known TCP port. A TCP port can be marked as a *replicated port* by means of the system call

```
int setportopt(port, mode,
               detector-parameters);
```

where `port` defines the port number, `mode` is used to indicate whether the replica binding to the port is a primary server or a backup, and `detector-parameters` defines the behaviour of the failure detector for this port. Whenever a socket for a replicated port is created, the kernel marks the protocol control block accordingly and initializes the failure detector. The replica management protocol contacts the nearest redirector about inserting the newly created socket into the acknowledgement channel, which is further described later.

4.2. Redirectors

The redirectors in HYDRANET-FT act as described in Section 3 for services that are simply replicated. In ad-

dition, the redirector maintains information about fault-tolerant services and their replicated ports: For each service on a replicated port, the redirector maintains the location of the primary server and of all the backup servers. Whenever a datagram matches a pair of destination IP address and port number in the redirector table, it is encapsulated and tunnelled to the appropriate hosts, with one copy going to the primary server and one copy to each backup server. Redirectors are made aware of the existence of the *primary server*, and the replicas designated as *backup servers* by means of the *replica management protocol*, similar to HYDRANET. The management of replicas in HYDRANET-FT is further described in Section 4.4.

One goal in HYDRANET-FT was to keep the operation within redirectors as simple as possible. It will become evident below that redirectors take no part in providing fault-tolerant message delivery to servers or to clients, except for providing a simple form of non-reliable multicast and for acting as an access point for the replica management protocol. In addition, there is no need for redirectors to handle messages directed from servers to clients, which makes routing of return-paths more flexible.

4.3. Atomicity, Message Ordering, and Virtual Synchrony

Atomicity and ordering of message delivery is maintained with the help of a one-way acknowledgment channel between the backups and the primary. Backups are daisy-chained along this channel as described below, starting at the primary. While all the replicas (primary and backups) receive the TCP data (data and flow control information) from the client, only the primary responds to the client. Instead of responding, backups pass the control flow information only to the previous server up the chain, with the first backup passing that information to the primary. For convenience, we have S_0 denote the primary server, and $S_1, \ldots, S_N$ the N backup server, as they appear in the daisy chain.

Figure 3 illustrates this with a simple system of a primary server S_0 with two backup servers S_1 and S_2. Packets from clients to the replicated server are detected at the redirector, which sends one copy of the packet each to the three servers S_0, S_1, and S_2. Only the primary server, S_0, responds to the client. The backup servers, S_1 and S_2, do not respond to the client directly. Instead, they send all the relevant acknowledgement information along the acknowledgement channel to the previous server, in this case from S_2 to S_1, and from S_1 to S_0.

When a backup server S_i is ready to send a TCP packet (be it a data packet, or a packet with only flow control information) it sends to the previous server S_{i-1} the two flow control fields in the TCP header of the packet: the SEQUENCE NUMBER of the packet, which identifies the position in the server's byte stream of the data in the packet, and the ACKNOWLEDGEMENT NUMBER, which is the number of the byte that the server expects to receive next.

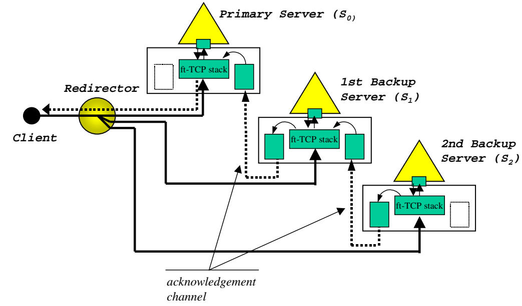

Figure 3. Example of a Replicated Server with one Primary and two Backup Servers.

Atomicity. This is achieved by synchronizing the depositing and retrieving of data to and from the socket buffer using these two numbers. Atomicity of message delivery from the client to the servers is maintained by allowing each server S_i to deposit a Byte k from the data stream coming from the client into the socket buffer only after an acknowledgement message has been received from S_{i+1} with an ACKNOWLEDGEMENT NUMBER larger than k. (The last backup server in the chain, S_N, is free to immediately deposit the data.) In this fashion, it is guaranteed that Server S_i puts incoming data from the client into the socket buffer only after all servers $S_{i+1}, \ldots, S_N$ have done so.

Once S_i has deposited the data in the socket buffer, it forwards the flow control information along the acknowledgement channel to server S_{i-1}. Once the primary server S_0 receives the data and the acknowledgment information for that data from backup server S_1, it replies to the client, and so closes the familiar TCP flow-control loop.

In a similar fashion, the outgoing data from the servers is synchronized with the help of the SEQUENCE NUMBER FIELD forwarded along the acknowledgement channel. More precisely, no server S_i is allowed to send Byte k before a SEQUENCE NUMBER larger or equal than k has been received from server S_{i+1}. Again, the last backup server, S_N is free to immediately send out data from its socket buffer. Note again that outgoing packets of backup servers are not actually sent to clients. The acknowledgement information is stripped from the packet header and forwarded to the predecessor server along the acknowledgment channel. The contents of the packet is then discarded.

We note that the acknowledgment channel does not enforce full synchronization between the servers. At any given point in time, the last backup server S_N can receive (and send) up to a window full of data ahead of the primary

server. As it is the primary server to communicate with the client, this lack of synchronization has no effect in case of failure. As this all is handled by the TCP error control mechanism. If the primary fails, for example, the backups will experience transmissions after the recovery of data previously received. TCP gracefully discards this data.

Message Ordering. Message ordering within a single connection is guaranteed by TCP sequence numbering. Ordering across connections to the same replicated TCP port is assured if the acknowledgement channel provides in-order message delivery. In the current implementation we use a kernel-to-kernel UDP connection for the acknowledgement channel, trading low overhead against lack of ordering across connections and against client re-transmissions if packets on the acknowledgement channel are lost.

Failure Detection and Virtual Synchrony. If a server fails to receive a packet, the flow control loop is broken, and the client re-tranmits. If this re-transmission was caused by a lost packet on a server, say Server S_i, the message delivery to servers simply picks up where it was interrupted, that is, with Server S_i receiving the data, and perhaps sending acknowledgment information up the acknowledgment channel. If the problem persists, on the other hand, the client keeps re-transmitting. Repeated re-transmissions are detected at the servers. After some number of re-transmissions have been detected, any server can initiate a reconfiguration of the set of replicas. Setting the detection threshold in number of re-transmissions before action is taken is a trade-off between detection latency and chance of "false positives". In addition, the thresholds should be high enough to not interfere with TCP's own congestion control mechanism, which for example initiates a slow-start recovery from link congestion after detecting a triple acknowledgment.

In general, this effective failure detection mechanism, in combination with atomicity of message delivery as described above, allows for a virtually synchronous reconfiguration of server after failures [15].

4.4. Replica Management Protocol

The architecture of the management protocol in both HYDRANET and HYDRANET-FT is patterned after the route management infrastructure for IP, with management daemons running on all HYDRANET hosts and the redirectors. The management daemons interact with each other using UDP for idempotent operations and a form of reliable UDP for the message exchanges. The daemons interact with the kernel on the local machine through raw routing sockets. The general operation of the replica protocol are itemized as follows:

Creation of primary server: When a server program binds to a replicated port on the primary server, an entry is created in the local kernel table to notify the host. The management daemon on the primary server also sends a message to the redirector informing it of the new primary. The redirector updates its kernel routing tables.

Creation of backup servers: In addition to the setup procedure described above, when a backup server is created, the redirector looks up the routing tables and sends to the backup the IP address and port of the server ahead of it in the acknowledgement channel.

Deletion of primary server: When a primary server needs to voluntarily leave the HYDRANET-FT setup, the management daemon on the server informs the redirector. If the server is a primary, the redirector designates the backup immediately following the primary in the acknowledgement channel as the new primary and sends a message to it to reconfigure itself as a primary.

Deletion of backup server: When a backup server leaves, it is eliminated from the acknowledgement channel.

Reconfiguration after a failure detection: When a server detects a failure, it informs the redirector. Reconfiguration consists of two steps: First, the failed server needs to be identified. As a failure partitions the acknowledgement channel, identification is simple. The failed server must then be "shut down" by eliminating it from the set of replicas and removing it from the acknowledgement channel. If the failed server was the primary server, a new primary server needs to be designated as part of this deleting procedure. The reconfiguration is trivial in the case of a single-backup system, in which the backup can immediately take over as primary server.

5. Experiments

We measured the performance impact of our BSD implementation of HydraNet-FT on a small testbed, which, for measurement purposes, consists of two Pentium/120 PC and two 486 PCs. (In this setup with antiquated equipment we purposely used slow machines to measure the effects of bottlenecks.) We did measurements with `ttcp` to determine the overhead, in terms of reduced throughput, in redirectors and host servers. We set one 486 PC to act as the redirector and the two Pentiums as Primary and Backup. Another 486 PC is client. We compared the sustained bandwidth of TCP for the following four series of measurements. (For the measurements, we turned off buffering of small segments at the TCP sender, preventing it from batching multiple small segments into a segment of MTU size.)

Clean: All machines run unmodified system software. No redirection happens and no services are replicated. These measurements act as baseline for performance comparison.

No Redirection: The routers and the receivers run the *HydraNet-FT* modified system software. There is no redirection.

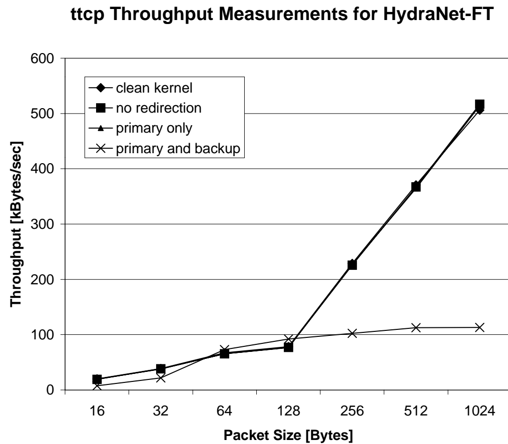

Figure 4. Throughput measurements for HydraNet-FT

To Primary only: Configuration is same as above. The packets, however, are destined to a port on a non-existent host with a replica running as Primary server on the host server. There are no backup servers. These experiments illustrate the penalties caused by redirection.

To Primary with single Backup: In this configuration, the redirector multicasts packets to the Primary and the Backup server. The resulting throughput indicates the performance of the HYDRANET-FT protocol with a single backup.

Figure 4 depicts the throughput measurements for the four cases. The general trend indicates that throughput rises as the packet size increases. This is to be expected, since for small packet sizes the TCP/IP header processing proves to be a significant overhead. Also, beyond packet size of MTU, the throughput drops again. This is due to the fragmentation of packets. From the graphs, it is evident that the HYDRANET-FT when operating in Fault-tolerant mode with a server and a backup offers a throughput that is not unreasonably lower than that offered by a TCP connection between a single server and client (as in case of the clean kernel). A closer examination of the experimental data indicates that most of the performance hits happen in form of timeouts at the client, with successive re-transmission because of packets being dropped at the primary. In this case it is the lengthy timeout, not the re-transmission, which affects the performance. This is strictly due to our conservative modification of the TCP/IP stack, and we are confident that the throughput can be further improved by eliminating unnecessary timeouts and retransmissions of this form.

6. Conclusion

As long as the Web was used mostly as backbone for data dissemination and relied on stateless servers, the idempotency of HTTP allowed to handle server failures by simple replication of servers and provision for some means (naming-based or other) to redirect the request around the failed server. Over the years, however, the Web has developed into the infrastructure of choice for a large variety of often rather complex applications. Some of these are transaction based, for example many e-commerce systems, and have servers maintain much state. Some others may have long-lasting sessions, such as media streaming or data feeders, again causing servers to keep state. Plain service request redirection is not sufficient to recover from server failures for these classes of applications.

Existing approaches for robust applications over conventional hardware typically rely on messaging subsystems with some support for reliable multicasting, and typically require that client programs have these capabilities as well. In this paper we present an infrastructure for fault-tolerant service replication that is fully transparent to the client. It allows to replicate a service across several host servers. When a particular server becomes inaccessible, we rely on the reconfiguration capability of routers and redirectors to appropriately redirect requests to the remaining replicas of the service.

For TCP-based services, we take advantage of the flow-control and error-control mechanisms in TCP to provide an accurate failure estimator (through monitoring of client retransmissions) and ordered atomic multicast to the service replicas (through the window-based flow control at the two TCP end points). The result is a very natural extension of the service model provided by TCP: while TCP guarantees reliable communication as long as there is a path between client and server, HYDRANET-FT guarantees reliable communication as long as there is a path between the client and *at least one* operational server. Providing this level of fault tolerance at TCP level is particularly interesting for two reasons: First, it does not affect clients, thus allowing for deployment of fault-tolerant services with a large population of existing clients, for example Web-browsers. Second, it builds on a familiar and simple-to-use programming interface (TCP streams) both on client and server side.

We have implemented this approach to fault-tolerant service replication as part of HYDRANET, an extension of the BSD process management and TCP/IP protocol processing, which allows to dynamically install replicas of server processes across an internetwork. Requests to replicated services are redirected to the replicas, and the modified TCP protocol software on the replicas takes care of atomic, ordered delivery of packets to the socket buffers of the server application. The same mechanisms also ensures atomicity of many-to-one message delivery from servers to clients. A suite of experiments indicates that the performance (we show results of throughput measurements on the HYDRANET testbed) of this approach is high.

A number of issues need to be further investigated in the context of fault-tolerant services on the proposed infrastructure. We provide ordered atomic delivery to and from the transport-level service access points of the replicated service (i.e. the TCP socket buffers of the replica server processes). Unfortunately, this is not sufficient to ensure event ordering across the servers. While message delivery is indeed ordered, thread scheduling and dispatching may cause messages to be processed in different order across servers. This problem is not specific to our approach. Rather, it is inherent in any approach that does not guarantee atomicity of message delivery *and* of event processing. This is a common problem in fault-tolerant ORBs over fault-tolerant messaging subsystems, for example [15].

While we provide effective procedures for failure detection and for fail-over from the primary to the backup servers, no appropriate mechanism is in place to allow for transparent re-commissioning of recovered servers. This is particularly important for temporary failures due to congestion in the network or at the server. The server recovery procedure must ensure the required level of consistency of states at application level. In addition, in accordance with our requirements for client transparent failure and recovery, mechanisms must be put in place to transfer the current state of the TCP connection(s) from the backup to the primary. We are currently investigating the latter issue.

References

[1] Y. Amir, D. Dolev, S. Kramer, and D. Malki. "Transis: A communication subsystem for high availability." *Proceedings of the 22nd Annual International Symposium on Fault-tolerant Computing*, pp 76-84, July 1992.

[2] D. Andresen, T. Yang and O.H. Ibarra. "SWEB: Towards a Scalable World Wide Web Server on Multicomputers." *Proceedings of the IPPS'96*, pp 850 – 856, April 1996.

[3] A. Bhide, E. Elnozahy, S. Morgan, A. Siegel. "A Comparison of Two Approaches to Build Reliable Distributed File Servers." *Proceedings of the 11th International Conference on Distributed Computing Systems*, pp 616-623, 1991

[4] K. P. Birman and R. van Renesse, eds, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press, 1994.

[5] N. Budhiraja and K. Marzullo. "Highly-Available Services using the Primary-Backup approach." *Second Workshop on Management of Replicated Data*, pp 47-50, 1992

[6] H. Chawla, G. Dillon and R. Bettati, "HYDRANET: Network support for scaling of large-scale services," *Journal of Network and Computer Applications*, to appear.

[7] P.Y. Chevalier. "A Replicated Object Server for a Distributed Object-Oriented System," *Proceedings, Eleventh Symposium on Reliable Distributed Systems*, pp 4-11, 1992.

[8] P. Chundi, R. Narasimhan, D. Rosenkrantz, and S. Ravi. "Using Active Clients to Minimize Replication in Primary-Backup protocols." *Proceedings, 1996 IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pp 96-102, 1996

[9] S. Deering and D. Cheriton. "Multicast Routing in Datagram Internetworks and Extended LANs." *ACM Transactions on Computer Systems*, 8 (2), 85-110.

[10] R. Friedman and R. van Renesse. "Strong and weak virtual synchrony in Horus." *Proceedings, Fifteenth Symposium on Reliable Distributed Systems*, pp 140-149, 1996.

[11] R. Gamache, R. Short, and M. Massa. "Windows NT Clustering Service." *Computer Magazine*. Vol. 31(10), Oct 1998, pp 55-62.

[12] M. Hayden. "The Ensemble System." Cornell University Technical Report TR98-1662, January 1998.

[13] S. Landis and S. Maffeis, "Building Reliable Distributed Systems with CORBA," *Theory and Practice of Object Systems*, New York: John Wiley, April 1997.

[14] J. Lyon. "Tandem's Remote Data Facility." *COMPCON Spring 90, Digest of Papers*, pp 562-567, Feb 1990.

[15] S. Maffeis and D. Schmidt. "Constructing Reliable Distributed Communication Systems with CORBA." *IEEE Communications Magazine*, February 1997, pp 56-60.

[16] S. Mishra, L.L. Peterson, and R.D. Schlichting. "Consul: A Communication Substrate for Fault-Tolerant Distributed Programs." *Distributed Systems Engineering Journal*, 1, 2, (Dec. 1993).

[17] L. E. Moser, P. M. Melliar-Smith, A. Agrawal, R. Budhia, A. Lingley-Papadopoulos, and T. Archambault. "The Totem System." *Proceedings of the Twenty-Fifth International Symposium on Fault-tolerant Computing*, pp 61-66, 1995.

[18] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, "Consistent Object Replication in the Eternal System," *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998).

[19] L. Wang, W. Zhou. "Primary-Backup Object Replications in Java." *Technology of Object-Oriented Languages, Proceedings*, pp 78-82, 1998.

[20] H. Zou and F. Jahanian. "Real-Time Primary-Backup (RTPB) Replication with Temporal Consistency Guarantees", *Proceedings, 18th International Conference on Distributed Computing Systems*, pp 48–56, 1998.