Network Working Group
Internet-Draft
Expires: May 16, 2007

J. Loughney, Ed.
Nokia
A. Silverton, Ed.
Motorola
M. Stillman
Nokia
Q. Xie
Motorola
R. Stewart
Cisco

Nov 16, 2006

Comparison of Protocols for Reliable Server Pooling
draft-ietf-rserpool-comp-11.txt

Status of this Memo

Copyright Notice

Abstract

This document compares protocols that may be applicable for the

Reliable Server Pooling problem space. This document discusses the usage and applicability of these protocols for the Reliable Server Pooling architecture.

Table of Contents

1. Introduction

1.1 Overview

In creating a solution to provide reliable server pools [1], there are a number of existing protocols, which appear to have similar properties as to what RSerPool is trying to accomplish. This document discusses the applicability of these protocols in meeting the requirements of Reliable Server Pooling [2].

This study does not intend to be complete, rather intends to highlight several protocols which working group members have suggested.

1.2 Terminology

This document uses the following terms:

Operational Scope: The part of the network visible to pool users by a specific instance of the reliable server pooling protocols.

Pool: A collection of servers providing the same application functionality. Also called a Server Pool.

Pool Handle: A logical pointer to a pool. Each server pool will be identifiable in the operation scope of the system by a unique pool handle or "name". Also called a Pool Name.

Pool Element: A server entity having registered to a pool.

Pool User: A server pool user.

Handle-Space: A cohesive structure of pool names and relations that may be queried by an internal or external agent.

ENRP Server: Entity which is responsible for managing and maintaining the handle-space within the RSerPool operational scope.

1.3 Abbreviations

DA: Directory Agent in SLP.

DPE: Distributed Processing Environment.

CORBA: Common Object Request Broker Architecture.

OMG: Object Management Group.

ORB: Object Request Broker.

PE: Pool Element.

PU: Pool User.

SA: Service Agent in SLP.

SLP: Service Location Protocol.

UA: User Agent in SLP.

TTL: Time to live in DNS.

2. Relation to Other Technologies

This section is intended to discuss the applicability of some existing solutions with regards to Reliable Server Pooling requirements [2]. The protocols discussed have been suggested as possibly overlapping with the problems space of RSerPool.

2.1 CORBA

Often referred to as a Distributed Processing Environment (DPE), CORBA was mainly designed to provide location transparency for distributed applications. CORBA's distribution model encourages an object-based view, i.e., each communication endpoint is normally an object.

CORBA has a number of variants, such as fault-tolerant CORBA, Real-time CORBA, etc. CORBA has been used in a number of situations, for example, Real-time CORBA has been used in fighter aircraft and weapon systems. Additionally, CORBA has been implemented in a wide range of devices, from attack submarines to Palm Pilots - the MICO open source ORB has been ported to the Palm Pilot, and the client- only application is 45 KB in size.

CORBA, a DPE, sits above the communications layer that is the purview of most IETF work, specifically, RSerPool. In a conceptual model of a middleware stack for highly available clustering, CORBA is considered an application service and not a messaging or clustering service. A distributed application may utilize a CORBA ORB for location transparency at the application layer, and the ORB may in

turn utilize RSerPool for its communications layer. For example, a real-time CORBA ORB such as Tau, would benefit from having its interfaces extended to take advantage of RSerPool concepts.

2.2 DNS

This section will answer the question why DNS is not appropriate as the sole solution for RSerPool. In addition, it highlights specific technical differences between RSerPool and DNS.

During the 49th IETF December 13, 2000 plenary meeting Randy Bush presented a talk entitled "The DNS Today: Are we overloading the Saddlebags on an Old Horse?" This talk underlined the issue that DNS is currently overloaded with extraneous tasks and has the potential to break down entirely due to a growing number of feature enhancements.

RSerPool and DNS are protocols with very different objectives. RSerPool is designed to provide a range of services up to the point of relieving an application of the overhead of maintaining a session with an active server. DNS was not intended to handle such a range of functions. DNS may, however, be able to handle some of the lower range of RSerPool functionality.

One requirement of any solution proposed by RSerPool would be to avoid any additional requirements for DNS in order to support Reliable Server Pooling. Interworking between DNS and RSerPool will be considered so that additional burdens to DNS will not be added.

2.2.1 Requirements

Any solution for RSerPool should meet certain requirements [2]. These requirements are discussed below in relation to DNS.

> "Servers should be able to register to (become PEs) and deregister from a server pool transparently without an interruption in service.
>
> The RSerPool mechanisms must be able to support different server selection mechanisms. These are called server pool policies.
>
> The RSerPool architecture must be able to detect server failure quickly and be able to perform failover without service interruption.
>
> Server pools are identified by pool handles. These pool handles are only valid inside the operation scope. Interoperability between different namespaces has to be provided by other

mechanisms."

2.2.2 Technical Issues

This section discusses the relationship between DNS and the requirements for RserPool.

2.2.2.1 Host Resolver Problems

A major issue that prevents the use of DNS as part of the RSerPool solution is the architecture of host resolvers. These are stub resolvers - which means that they require their local DNS servers to do recursion for them.

In turn, this implies that setting TTL low or 0 will dramatically increase the load not only on the authoritative DNS servers - but also on these third party servers.

A secondary effect of this is that the authoritative DNS will not know the IP address of the DNS client - only the IP address of the local DNS. This affects the ability to do global load balancing correctly.

There is no way to get around these issues unless one requires all hosts to be full resolvers. Putting full resolvers on newer hosts isn't sufficient because the issues would still exist for all the legacy systems, which will form the bulk of the host population for years to come. The solution is not to use third party servers.

Additionally, if the client can contact the server directly, then the server knows the real IP address of the client. Since there is no third party involved, the caching TTL can be set as low as desired (even to zero). That will increase load on the server, but nowhere else.

Finally, DNS is based on a recursion. This recursion presents certain difficulties for RSerPool. Even if a host resolver is not a stub resolver, it has to go to another full resolver where 2 possibilities exists: either the mapping name-IP address is found or it has to do another recursive resolution of the name, staring from that intermediate resolver, until there is a cache hit in one of the intermediate resolvers or it is resolved by its root resolver (or home DNS server).

This process of recursion means that there is no end-to-end communication between the host and its server where the name-to-IP mapping resides. That also means that a lot of timers are running in

intermediate systems. Any updating of the transient status of the pool element or of the pool may need to be propagated through the DNS.

2.2.2.2 Dynamic Registration

Registration / de-registration of servers is needed. It can be done with DNS by NOTIFY/IXFR. However, frequent updates and replication are incompatible. This is not a DNS problem per se, but it has an effect on DNS as it is deployed.

RSerPool must allow software server entities (i.e., PEs) to register themselves with a name server dynamically. They can also de-register themselves for purposes of preventative maintenance or can be de-registered by an ENRP server that believes the server entity is no longer operational. This is a dynamic approach, which is coordinated through servers in the pool and among RSerPool ENRP servers.

2.2.2.3 Load Balancing

RFC 2782 [3] itself points out some of the limitations of using DNS SRV for load balancing between servers.

> Weight is only intended for static, not dynamic, server selection. Using SRV weight for dynamic server selection would require assigning unreasonably short TTLs to the SRV RRs, which would limit the usefulness of the DNS caching mechanism, thus increasing overall network load and decreasing overall reliability.

Based on this, DNS can only really support stochastic load balancing, redirecting clients to servers randomly as various caches in various resolvers expire at random (although small) intervals. DNS offers excellent network scalability but poor control over load balance.

As mentioned previously, the issue of doing DNS-based dynamic load balancing on short time scales will have impacts on third parties, due to the presence of stub resolvers.

2.2.2.4 Heartbeating & Status Monitoring

RSerPool working group has agreed that one of its main design goals for RSerPool is "...performance for supporting real-time applications", as reflected in RFC 3237 [2]. An example of such real-time applications would be the IP-based call control applications in a 3G cellular network.

To achieve this goal, it is felt critical that RSerPool monitors the state of each server entity on various hosts on a continual basis and

collects several state variables including up/down state and current load. If a server is no longer operational, eventually it will be dropped from the list of available servers maintained by the ENRP server, so that subsequent application name queries will not resolve to this server address.

DNS does not incorporate an application layer heartbeat. Heartbeating would dramatically boost traffic levels, and given the unavoidable third party dependencies of DNS, the resulting loading would be unacceptable. It is passive in the sense that it does not monitor or store information on the state of the host such as whether the host is up or down or what kind of load it is currently experiencing.

It is not entirely impossible to make DNS utilize the assistance of an external heartbeat function/protocol for this monitoring purpose. However, to achieve the degree of real-time performance RSerPool is seeking, one would most likely need a tight coupling of this external function to the DNS operation. This in turn would likely result in substantial modification of the existing DNS, which is what we want to avoid.

2.2.2.5 Name/Address Resolution Granularity

The technical requirement for DNS name/address resolution is basically that given a name, find a host associated with this name and return its IP address(es). In other words, in DNS we have the following mapping:

```
Name     ; a host machine

-to-

Address  ; IP address(es) to reach the host machine
```

The technical requirement for RSerPool name/address resolution is that given a name (or pool handle), find a server pool associated with this name and return a list of _transport_ addresses (i.e., IP address(es) plus port number) for reaching a set of currently operational servers inside the pool. In other words, in RSerPool we have the following mapping:

```
Name     ; a handle to a server pool

-to-

Address  ; transport address(es) (i.e., IP plus port number) to
         ; reach a set of functionally identical server entities
         ; (software processes). These software entities can be
         ; distributed across multiple host machines.
```

Without significant extensions, the current DNS would have difficulty achieving this type of name mapping.

2.2.2.6 Lack of Support for Real-time Fault Detection and Recovery

Even if we could somehow overcome the aforementioned shortcomings of DNS in terms of providing the name resolution service to RSerPool, we still would not have the support for real-time fault detection and recovery (i.e., failover) which is a key requirement in RSerPool.

To meet this requirement, a mechanism would need to be in place that would detect the unreachability of a message recipient and re-direct or re-route a user message to an alternate recipient in the same destination pool in real-time or semi-real-time. DNS currently contains no such mechanism.

2.2.2.7 Lack of Support for Redundancy Models

Server pooling as defined in RSerPool requires support for different redundancy arrangements or models depending on the needs of the specific application. Commonly used models in practice includes N+M, N-active, etc. These models basically define how a PE behaves when another PE in the same pool fails and it is often critical for the application to have full control over this behavior of each PE in the pool. Without major extensions, it seems difficult for DNS to support such redundancy models.

2.3 Dynamic Delegation Discovery System (DDDS) and URI

In this section, we discuss the difficulties for RSerPool to make use of DDDS and URI as building blocks for its distributed pool handle database (i.e., RSerPool handle-space).

RFC 3401 [5] defines DDDS as an abstract algorithm for applying dynamically retrieved string transformation rules to an application-unique string. DDDS has been found useful for URI Resolution, ENUM telephone number to URI resolution, and the NAPTR DNS resource record.

As stated in [5], DDDS is "used to implement lazy binding of strings to data, in order to support dynamically configured delegation systems. The DDDS functions by mapping some unique string to data stored within a DDDS Database by iteratively applying string transformation rules until a terminal condition is reached."

In order to discuss the applicability of DDDS/URI in RSerPool, we need first talk about some fundamental characteristics of pool handles or names in RSerPool.

It is important to note that, handles or names in RSerPool are meant to identify pools comprising of _generic_ communications nodes. Those nodes in reality will be some kind of servers - service applications that runs on some networked machines. However, it is very important to note that the working group has never had the intention to go beyond the "server of generic IP applications" in its pool definition. Nor has it seen the need to categorize the types of the service applications for the purpose of RSerPool. This is fundamentally different from the assumption behind URI as well as the Service Location Protocol (to be discussed below).

With the above noted, here are some additional characteristics of RSerPool handles/names:

1. RSerPool handles have only local significance, i.e., there is no requirement for pool handles to be globally unique.

   This is because the first tier of applications we envision for RSerPool is those tightly coupled local systems that can use RSerPool to make its components highly available. For example, a 3G radio access network that contains charging server, call controller, media server, etc. where RSerPool can make those currently singleton elements into pools and thus gain high availability. This type of local systems can be as compact as a bunch of server blades located in a single high performance chassis or a group of closely located boxes in a central office or in a few closely located buildings. The use of RSerPool in such scenarios can be totally transparent to the outside world.

   For example, a SIP phone may be talking to a softswitch without knowing that the call control elements inside the softswitch are a bunch of RSerPool-enabled pools. In such cases, the pool names has no need to be globally unique (there is even no need for the outside to know they exist). They only need to be unique within the softswitch itself.

   We also have considered the possibility of supporting larger scale (even global) deployment cases of RSerPool. In the

requirement, we indicate we want RSerPool to be able to do that but we have made it clear that RSerPool will not be able to do that by itself. Instead, it will rely on existing external infrastructures (e.g., DNS, possibly URN/DDDS) to bridge locally scoped RSerPool clouds into a larger scale deployment.

2. RSerPool handles have no need for supporting any structure/ syntax.

   As merely locally significant identifiers for distinguishing pools of generic communication nodes, we consider adding structure/syntax to RSerPool handle definition will buy us nothing but will have real negative impact on the performance and increase the implementation complexity. The only recommend we have made so far is to use NULL-terminated ascii string for the pool handles. This seems to meet our needs nicely.

3. RSerPool handles are relatively dynamic.

   We consider that the pools may change relatively frequently; they may come and go as the system re-adjusts its capacity or configuration. We do not envision the handles to be long lived.

4. RSerPool handles are not for human readers.

   Unlike UNIs (URN/URL), we do not envision RSerPool handles to appear in e-mails, web documents, etc. for human viewing. Probably the only case where RSerPool handles will be read by a human is in a log file or configuration file, just like other system configuration parameters.

Due to the aforementioned characteristics of RSerPool handles, we do not see the benefit for directly using URNs/URIs for RSerPool handles. The two rather fundamental requirements (per RFC 1737) that brought us URNs - the global scope and persistence - do not apply to RSerPool handles. Moreover, as mentioned above, we see little benefit of making RSerPool handles human-readable or parsable in free text.

In the future, there may be the possibility that URNs and the associated infrastructure (e.g., DNS, DDDS) to play a vital role when we start to consider wide area or global deployment scenarios for RSerPool. For instance, a SIP client device that is looking for certain network resource will start with a URN that is eventually resolved to a pool handle that is then passed to an RSerPool namespace server, which in turn will resolve the pool name to a list of reachable/routable transport addresses of the server instances.

Similarly, since RSerPool handles are not global and have no structure/syntax, we do not see that using DDDS inside RSerPool can bring meaningful benefits.

2.4 Service Location Protocol (SLP)

2.4.1 Introduction

SLP [4] is comprised of three components: User Agents (UA), Service Agents (SA) and Directory Agents (DA). User agents work on the user's behalf to contact a service. The UA retrieves service information from service agents or directory agents. A service agent works on behalf of one or more services to advertise services. A directory agent collects service advertisements.

The directory agent of SLP simply acts as a cache and is passive in this regard. The directory agent is optional and SLP can function without it. It is incumbent upon the servers to update the cache as necessary by reregistering. The directory server is not required in small networks as the user agents can contact service agents directly using multicast. Unicast queries to SAs are possible subsequent to the UA having discovered them. User agents are encouraged to locate a directory at regular intervals if they can't find one initially, otherwise they can detect DAs by listening passively for DA advertisements.

2.4.2 What to Use

Figure 1 shows how SLP might be realized to provide RSerPool endpoint name resolution (ENR) services:

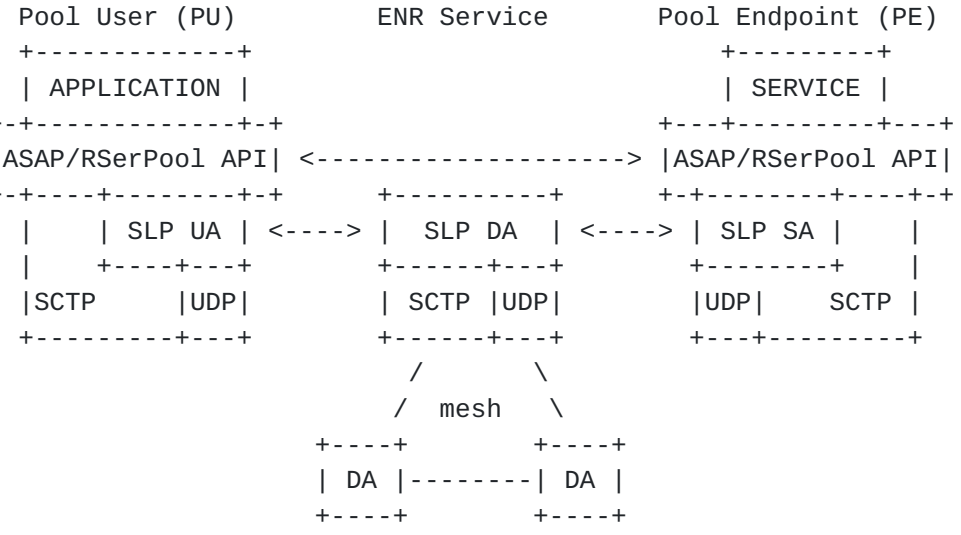

Figure 1: RSerPool entities employing SLP for ENR services

Notes:

- Each box constitutes a host (running a PU, PE or ENR server 'stack'), though one host could support more than one of these functions.
- As far as the Application is concerned, it is using a framework for exchanging messages with services reliably.
- As far as the Service is concerned, it is making itself available to a reliable server pool by interacting with the framework API.
- The ASAP/RSerPool API obtains endpoint name resolution data in a timely and robust manner and uses it to determine how to route PU requests to PEs.
- The ENR service function is performed using SLP. The PU employs a SLP UA to obtain information from a SLP DA.
- The PE employs a SLP SA to register information with a SLP DA. As the SLP SA is 'mesh-enhanced,' it only registers with one DA of this type (as long as it detects that this DA is alive & responsive & returns 'OK' results).
- The SLP DA is part of a mesh. It will forward PE state to other DAs in the mesh. For example, it will forward the registrations the SLP SA made on behalf of the PE on right of Figure 1.
- SCTP is used for communication between entities. Multicast UDP is used by SLP entities for active and passive discovery. While the RSerPool architecture cannot rely upon multicast mechanisms, it can profit from them when these are present in the network

SLPv2 will be needed, but SLPv2 alone does not fulfill RSerPool update requirements for timeliness. This is achieved through mesh-enhancements to the Service Location Protocol (mSLP) [6].

These enhancements make it possible for SAs to know of only a subset of all DAs. Mesh-enhanced SAs need only forward their registrations to only one mesh-enhanced DA. The mesh takes care of forwarding the message to the other DAs.

2.4.3 Summary of SLP Issues

A fundamental difference between SLP and RSerPool is that SLP is a protocol that focuses on the service level, while RSerPool is at the

communication level. More specifically, what SLP provides to its user is a mapping function from a name of a _service_ (e.g., b/w printing, color printing, faxing) to the location of the service provider, in the form of a URL string. The availability of the service provider is outside of the scope of SLP. How a service is accessible can be described by the SLP attribute list associated with the service URL. SLP is essentially a discovery protocol, not a transport protocol. Therefore, the granularity of SLP operation is at application service level.

In contrast, what RSerPool provides to its user is a mapping function from a communication destination name (i.e., a pool handle) to a set of routable and reachable transport addresses that leads to a group of distributed software server entities registered under that name. RSerPool has NO intention to understand or convey to its user what is the service (e.g., printing, faxing, document scanning) the named pool is providing at the application level. In other words, the responsibility of RSerPool is only to reliably deliver a user message to one of those server entities in the destination pool.

In theory, information such as transport addresses and their reachability could be represented in SLP attributes. Currently, mSLP would need changes, for example it was designed to scale to ~10 DAs not ~100 DAs. Additionally, SLP is currently designed to run on top of UDP and TCP. If SCTP support is needed, some additional specification work would be needed.

SLP security makes no attempt to address the confidentiality of data transmitted between SLP agents. To properly address this concern, SLP agents would need to establish secure communication with each other. This would be achieved through the use of IPSec Encapsulating Security Payload.

Server discovery, however, is something which SLP does well, and if used for RSerPool, this would be useful.

Other difficulties and shortcomings for using SLP to implement RSerPool include:

- Due to the fact that the resolution granularity of SLP is at the service level, it relies on a syntax rich scheme to define services (e.g., printers > color printers > color printers with 720+ resolution, etc). This implies that SLP implementation will need to perform syntax analysis, filtering, and parsing when a name is queried, and will need to dynamically search its name space to identify which entities are to be included in the response to a specific query. This type of complicated processing and searching for each query may severely limit the performance of

SLP in a real-time world which is a key requirement of RSerPool.

- Without major extensions, SLP will not be able to provide a solution for real-time or semi-real-time fault detection and recovery. This is partially because SLP is a discovery protocol, not a communication protocol.

2.5 L4/L7 Switching

2.5.1 Introduction

This section discusses L4 and L7 switching techniques and their relation to the RSerPool architecture [2]. Since these technologies are highly proprietary, it is difficult to discuss these techniques in a thorough manner.

In both cases, the deployment of these techniques is dependent upon the type of switching equipment deployed and breaks the end-to-end communication model required by RSerPool. These devices provide a specialized service intended to address a few network challenges, e.g., web caching and web cache load balancing, firewall load balancing, web server scaling, and streaming media load balancing. They are not robust methods for providing network reliability or highly reliable and highly available location transparent server clustering as required by RSerPool.

The following sections will provide an overview and example of each technique and an accounting for key RSerPool architectural requirements not met. See Section 3 for a more detailed accounting of requirements compliance.

2.5.2 L4 Switching

L4 devices make switching decisions based on the TCP or UDP port numbers of the packet in transit.

2.5.2.1 Example

Web caching is an example of L4 switching. The topology requires the introduction of an L4 capable switch in series with an existing network connection and L2/L3 switch. This is of particular use to web cache configurations where, for example, all traffic destined for port 80 (HTTP) could be redirected to a web cache or distributed by the switch across a number of web caches to achieve load balancing. The L4 switch can react to a failed cache and cease to send traffic to that device by automatically detecting that it is unreachable. This is all accomplished without any configuration on a client

device.

An equally compelling use for this type of switching is load balancing across firewalls. If the firewalls are employing stateful packet inspection for TCP connections, then the L4 switch must track which packets belong to which connection and see that all such packets are switched to the same firewall.

An L4 switch is incapable of differentiating between packets containing cacheable objects and non-cacheable objects, therefore, L7 devices, which look inside packets, are deployed where such determinations must be made. In general, anytime that knowledge of the application level data is required to make a switching decision, L7 devices must be deployed.

2.5.2.2 Technical Issues

The more general behavior of L4 switching, redirecting traffic based on destination UDP or TCP ports, is similar to a function provided by RSerPool. Where it differs in this regard is that L4 switching is dependent upon the network infrastructure and not peer-to-peer or end-to-end as required by RSerPool.

L4 switching meets the requirement of forwarding to active elements only, as a switch will detect unreachable PEs, but does not provide for the necessary registration and deregistration of PEs or resolution by name. L4 switches require the manual configuration of access control lists to determine switching behavior. This is achieved in RSerPool by more flexible means and without any dependence on specialized network equipment.

Most of the features of ASAP [7] and ENRP [8] are not met by a device employing L4 switching techniques. See the comparison table in Section 5.

2.5.2.3 Security Issues

It is not clear that L4 switching introduces any new security concerns. In fact, in a two-port security model, where secure RSerPool services are provided on one port, and similar, but insecure services, on another, L4 switching could be used to direct traffic to a secure or insecure PE or ENRP server as necessary.

2.5.3 L7 Switching

As previously mentioned, L7 switching was developed to differentiate between the type of objects being directed by network switches. In the L4 case, the devices cannot differentiate between the types of

data, only the destination of the packets containing that data. L7 switches look at the application layer of a packet in transit to determine what type of object is contained within.

2.5.3.1 Example

For an L7 switch to do this, it is necessary to intercept data midstream. In the case of HTTP, which is carried over TCP, the L7 switch must break the TCP handshake when a new request is made to the server attached to the switch. This process begins during the initialization of the TCP connection and before the higher level protocol, i.e., HTTP, sends its request. The switch acts as the server during the TCP SYN, SYN ACK, ACK handshake between that server and the client. Once the HTTP request is issued by the client and the switch decides that this is non-cacheable data that should be directed to the server as opposed to a web cache, the L7 switch sets up a second connection with the actual server through an additional three-way handshake. The switch will forward the client's request to the server and for the duration of this connection, must graft the client-switch and switch-server connections together by modifying IP addresses and TCP ports on the fly. Cacheable data is handled similarly, but is redirected to groups of web caches as opposed to the web servers.

2.5.3.2 Technical Issues

It is not clear that L7 switching adds anything, as a general mechanism, beyond what is provided by L4 switching, towards providing a sufficient RSerPool architecture.

While this technique can be very valuable as a means to scale web servers, it is apparent that it takes a significant amount of work on the part of the switch to realize these gains. The nature of this method also requires that for each type of application traffic handled, a custom software module must be written and added to the switch operating system. It is not known, due to the proprietary nature of these devices, if this can be done by the end user and/or added dynamically to deployed systems.

It may be possible to write custom modules for a switch, given the appropriate access to hardware and software, to provide some type of enhanced reliability in a controlled network. But, it is the aim of RSerPool to provide a general mechanism that is widely deployable and highly portable. L7 switching requires a significant amount of development to customize each of the endpoint switches to which PUs and PEs may be attached.

Also of concern is the compatibility of SCTP with L7 techniques. The

interception and subsequent splicing of sessions may nullify some of the inherent benefits of SCTP and certainly add additional and unnecessary complexity and latency to the transport layer. ASAP and ENRP along with the multi-homing and stream based behavior of SCTP provide more benefit than custom L7 switching would provide and at a significantly lower cost.

2.5.3.3 Security Issues

While L7 switches do provide some robustness to TCP-based DoS attacks directed at servers by requiring a proper three-way handshake, and they can be used to redirect encrypted traffic to certain servers better capable of processing that traffic, they may break the security model of RSerPool.

It may not be possible to make all the routing and switching decisions necessary to support RSerPool services without knowing more than just the destination address and port of a packet. The necessary extended attributes are not elements of L4 or L7 switching, but are instead, parameters of ASAP and ENRP. As the ENRP traffic is encrypted in RSerPool, the L7 devices would not be able to extract the necessary session layer data without becoming potential third party security liabilities.

2.5.4 Summary

The L4/L7 switching techniques, being network oriented services, are not able to provide the communications session oriented behavior required by RSerPool.

Adequate support for naming, as well as registration and deregistration services, is not provided by these devices. RSerPool requires a fault tolerant name service as well as the ability to register and deregister PEs in real-time. To accomplish this with L4/L7 switching, one would need to define a standard protocol to allow the switches to communicate amongst themselves and, perhaps, implement a co-resident name server on the switch.

The RSerPool communication model is broken as these mechanisms are deployed on switch hardware as opposed to end devices such as PEs, PUs, and ENRP servers. This implies a significant requirement for processing power and a lack of support for mobility. It is unlikely that one could or would build L4/L7 behavior into end devices and RSerPool requires peer-to-peer functionality.

The equipment needed to deploy such solutions can be an order of magnitude more expensive per port than a traditional L2/L3 device and oftentimes must be deployed in addition to L2/L3 hardware. It has

been observed that L4/L7 devices show poor performance when acting as an L2/L3 switch. This combined requirement for network infrastructure is not appropriate for RSerPool.

L4/L7 switching while clearly good in certain areas, lacks the ability to provide a robust framework for location transparent clustering capable of scaling in size and performance from web server or other Internet applications to real-time telecommunications infrastructure. There are a host of concerns with the ability of these techniques to meet critical RSerPool requirements in the areas of flexibility, adaptability, timing, security, etc. The amount of effort required to achieve RSerPool functionality across L4/L7 switches would amount to implementing RSerPool, as it is currently defined, on those very switches.

2.6 ASAP and ENRP

ASAP [7] and ENRP [8] are being developed in the RSerPool working group. Even though they are separate protocols, they are designed to work together.

2.6.1 ASAP

ASAP uses a name-based addressing model which isolates a logical communication endpoint from its IP address(es), thus effectively eliminating the binding between the communication endpoint and its physical IP address(es) which normally constitutes a single point of failure. In addition, ASAP defines each logical communication destination as a pool, providing full transparent support for server-pooling and load sharing. If multiple endpoints register under a the same name, a server pool is effectively created. It also allows dynamic system scalability - members of a server pool can be added or removed at any time without interrupting the service.

ASAP monitors the reachability of the Pool Elements in order to provide fault tolerance. To support real-time or semi-real-time fault detection and recovery, ASAP makes use of the peer reachability feedback from either the transport layer (such as SCTP) or the upper layer protocol and re-send (or failover) user messages to alternate PEs in the destination pool. Load sharing and redundancy model support is provided in ASAP at the message sender side. ASAP allows extensions to be made in the future to accommodate new load sharing policies and redundancy models.

ASAP supports the "keepalive" monitoring of PEs by the ENRP server and session failover, in which a set of application messages are defined as a "session" and ASAP provides best-effort transmission of all the messages in the "session" to the same PE in the destination

pool. For some classes of service, ASAP can provide failover for the remaining message in the "session" to an alternate PE if the first PE fails.

2.6.2 ENRP

ENRP defines procedures and message formats of a pool registry service (or name service) for storing, bookkeeping, retrieving, and distributing pool operation and membership information. It allows Pool Elements to be dynamically added, updated and removed from service. There are also protocol mechanisms for detecting and removing unreachable Pool Elements.

Within the operational scope of RSerPool, ENRP defines the procedures and message formats of a distributed, fault-tolerant registry service for storing, bookkeeping, retrieving, and distributing pool operation and membership information. This is to avoid the name service itself becoming a single point of failure in the system.

ENRP itself is dynamically scalable, meaning that new ENRP servers can be added and existing servers can be removed as needed. This feature can be used to achieve zero planned downtime upgrade of a system - a common requirement for many mission critical applications.

ENRP is not designed to scale Internet wide. It uses a flat name space model to gain performance. Other protocols, such as DNS could be used to bridge small ENRP name spaces to create a large scale name space.

3. Comparison Against Requirements

This section attempts to create a comparison table to compare the technologies and protocols which have been suggested as applicable to the RSerPool architecture.

	CORBA	DNS	SLP	ASAP ENRP	L4/L7
Robustness	Y	Y	Y	Y	Y
Failover Support	Y	P	P	Y	P
Communication Model	N	P	Y	Y	N
Processing Power	N	Y	Y	Y	N
Support of RSerPool Unaware Clients	N	Y	N	N	N
Registering and Deregistering	N	P	P	Y	N
Naming	Y	Y	Y	Y	N
Name Resolution only to Active Elements	Y	N	Y	Y	Y
Server Selection Policies	Y	P	P	P	P
Timing Requirements and Scaling	P	N	Y	Y	P
Scalability	N	Y	Y	Y	Y
Security - General	Y	P	P	P	P
Security - Name Space Services	P	P	P	P	N

Y = Yes, meets requirement
P = Partially meets requirement
N = No, does not meet requirement
N/A = Not applicable

Table1: Comparison Against Requirements

4. Security Considerations

This type of non-protocol document does not directly affect the security of the Internet.

5. IANA Considerations

This document creates no considerations for IANA.

6. Acknowledgements

The authors would like to thank Bernard Aboba, Erik Guttman, Matt Holdrege, Lyndon Ong, Jon Peterson, Christopher Ross, Micheal Tuexen and Werner Vogels for their invaluable comments and suggestions.

7. References

7.1 Normative References

[1] Tuexen, M., Xie, Q., Stewart, R., Shore, M., Loughney, J., and A. Silverton, "Architecture for Reliable Server Pooling", draft-ietf-rserpool-arch-09 (work in progress), February 2005.

[2] Tuexen, M., Xie, Q., Stewart, R., Shore, M., Ong, L., Loughney, J., and M. Stillman, "Requirements for Reliable Server Pooling", RFC 3237, January 2002.

7.2 Non-Normative References

[3] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[4] Guttman, E., Perkins, C., Veizades, J., and M. Day, "Service Location Protocol, Version 2", RFC 2608, June 1999.

[5] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS", RFC 3401, October 2002.

[6] Zhao, W., Schulzrinne, H., and E. Guttman, "Mesh-enhanced Service Location Protocol (mSLP)", RFC 3528, April 2003.

[7] Stewart, R., Xie, Q., Stillman, M., and M. Tuexen, "Aggregate Server Access Protocol (ASAP)", draft-ietf-rserpool-asap-09 (work in progress), June 2004.

[8] Xie, Q., Stewart, R., and M. Stillman, "Enpoint Name Resolution Protocol (ENRP)", draft-ietf-rserpool-enrp-09 (work in progress), July 2004.

Authors' Addresses

John Loughney (editor)
Nokia Research Center
PO Box 407
Nokia Group FIN-00045
Finland

Email: john.loughney@nokia.com

Aron J. Silverton (editor)
Motorola, Inc.
1301 East Algonquin Road
Mail Drop 2246
Schaumburg, IL 60196
USA

Phone: +1 847-576-8747
Email: aron.j.silverton@motorola.com

Maureen Stillman
Nokia
127 W. State Street
Ithaca, NY 14850
USA

Phone: +1 607-273-0724
Email: maureen.stillman@nokia.com

Qiaobing Xie
Motorola, Inc.
1501 W. Shure Drive, #2309
Arlington Heights, IL 60004
USA

Phone: +1 847-632-3028
Email: qxie1@email.mot.com

Randall R. Stewart
Cisco Systems, Inc.
8725 West Higgins Road
Suite 300
Chicago, IL 60631
USA

Phone: +1 815-477-2127
Email: rrs@cisco.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.