Version 1.0

Das *rsplib*-Projekt - Hochverfügbarkeit mit Reliable Server Pooling

Thomas Dreibholz (dreibh@exp-math.uni-essen.de)

In unserem Paper geben wir zunächst eine Einführung zu Reliable Server Pooling (RSerPool), danach stellen wir unsere Implementation vor - das rsplib-Projekt. Im Anschluß daran zeigen wir, wie basierend auf unserer Implementation eigene, hochverfügbare Anwendungen mit RSerPool entwickelt werden können – begleitet durch Code-Beispiele. Zum Abschluß stellen wir noch eine unserer Beispielanwendungen vor.

Die Geschichte von RSerPool

Die SIGTRAN-Arbeitsgruppe der IETF hatte es sich zum Ziel gesetzt, ein Protokollrahmenwerk für den Transport von Telefonsignalisierung über IP-Netzwerke zu standardisieren. Der in der Telekommunikationswelt verwendete Standard Signalling System No. 7, kurz SS7, stellt sehr hohe Anforderungen an Verfügbarkeit und Redundanz. Beispielsweise sollte eine Telefonvermittlungsstelle auch bei Ausfall von einzelnen Komponenten immer noch in der Lage sein, Notrufe vermitteln zu können. Wird statt eines herkömmlichen Telefonnetzes ein IP-Netz verwendet, muß natürlich eine eben solche Verfügbarkeit ebenfalls gewährleistet sein.

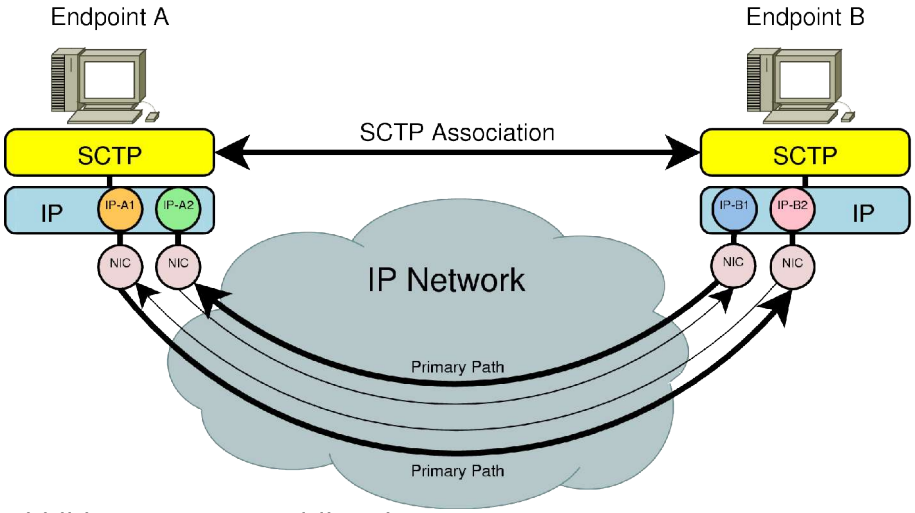


Abbildung 1: SCTP Multihoming

Aus diesen Überlegungen zur Verfügbarkeit heraus wurde das Transportprotokoll SCTP (Stream Control Transmission Protocol [16][20]) entwickelt, welches mittlerweile Standard ist und für alle gängigen Betriebssysteme verfügbar ist.

SCTP bietet insbesondere die Möglichkeit, Transportendpunkte an mehrere Netzwerkinterfaces zu binden: das sogenannte Multihoming. Abbildung 1 zeigt dies an einem Beispiel: Die Endpunkte A und B besitzen jeweils zwei Netzwerkinterfaces mit unterschiedlichen IP-Adressen (IPv4 oder IPv6; ein Mischen der Protokolle ist ebenfalls kein Problem) in unterschiedlichen Netzwerken. Sofern es im Internet disjunkte Pfade gibt kann einer der beiden Pfade ausfallen ohne die SCTP-Verbindung, Assoziation genannt, zu unterbrechen. Das SCTP-Protokoll sorgt automatisch bei Ausfall eines Pfades für eine Umschaltung. Zu beachten ist, daß aus Gründen der TCP-Freundlichkeit der Überlastkontrolle immer nur ein Pfad für die Benutzerdaten verwendet wird, der sogenannte Primary Path.

Mit der Add-IP-Erweiterung [14] von SCTP – welche von allen wichtigen SCTP-Implementationen unterstützt wird – ist es sogar möglich Transportadressen während der Assoziationslaufzeit hinzuzufügen oder zu entfernen. Das heißt, wird der Rechner mit einem neuen Netzwerk verbunden (z.B. wenn ein Wireless-AP erreichbar wird), so wird dieses automatisch für die Assoziation nutzbar.

Mit seinem 4-Wege-Handshake und dem sogenannten Verification Tag besitzt SCTP zudem einen relativ zuverlässigen Schutz vor Denial of Service-Angriffsarten für die TCP anfällig ist (z.B. SYN-Flooding oder das Senden von RST-Paketen). SCTP bietet zwar mit diesen Eigenschaften einen Schutz vor einer Vielzahl von Netzwerkproblemen, insbesondere auch Ausfall einzelner Links oder langer Konvergenzzeiten des BGP-Routings, allerdings schützt SCTP natürlich nicht davor, daß ein Endpunkt – insbesondere ein Server - selbst auch einmal ausfallen kann.

Eine einfache Methode zum Schutz vor Serverausfällen ist natürlich, Server einfach redundant in einem Pool vorzuhalten. Fällt nun einer der Server aus, kann einfach ein anderer verwendet werden. Ein solches Verfahren wird z.B. häufig für Webserver verwendet, ein System wie Linux Virtual Server (LVS) sorgt dabei für eine transparente Zuordnung von Anfragen zu einem funktionsfähigen Server des Pools. Systeme wie LVS sind allerdings für relativ spezielle Fälle ausgelegt: Fällt einer der Server einer Web-Server-Farm aus, sind dessen aktuell laufende Sitzungen (TCP-Verbindungen) unterbrochen. Der Client (z.B. ein Webbrowser) muß daher einen erneuten Download der Seite versuchen (Abort-and-Restart-Prinzip).

Während das Abort-and-Restart-Prinzip für Webserver sehr gut funktioniert, ist es sehr problematisch für lange andauernde Transaktionen (z.B. Data Mining in einer Datenbank). Auch für solche Anwendungen gibt es wieder spezielle Lösungen, die eine Wiederaufnahme unterbrochener Sitzungen ermöglichen. Allerdings ist es relativ ineffizient, für jede Anwendung ein eigenes System zur Redundanz zu entwickeln. Daher hat es sich die Reliable Server Pooling Working Group (RSerPool WG) der IETF zum Ziel gesetzt, ein allgemeines Protokollrahmenwerk zur Verwaltung von Server-Pools sowie Sitzungen mit diesen Pools in Form eines Session-Layers zwischen Transportprotokoll und Applikationsschicht zu entwickeln und zu standardisieren: Reliable Server Pooling, kurz RSerPool.

Was ist Reliable Server Pooling?

Anforderungen an RSerPool

Die an die zu definierende RSerPool-Architektur gestellten Anforderungen waren insbesondere folgende [23]:

Lightweight: Die RSerPool-Lösung soll nur wenig Ressourcen (z.B. CPU, Speicher) beanspruchen. Insbesondere soll es auch möglich sein, ein RSerPool-basiertes System mit Geräten wie Embedded Devices zu nutzen.

Echtzeitfähigkeit: Bei Anwendungen in der Telefonsignalisierung sind strikte Zeitvorgaben einzuhalten. Im Fall eines Komponentenausfalls muß innerhalb weniger Hundert Millisekunden wieder ein stabiler Systemzustand erreicht werden. RSerPool muß es deshalb ermöglichen, solche Zeitvorgaben auch einzuhalten.

Skalierbarkeit: Für Dienste wie die Verwaltung eines Distributed-Computing-Pools kann die Anzahl verwalteter Server auf mehrere Hundert oder gar Tausend steigen. Die RSerPool-Architektur muß daher entsprechend skalierbar sein; allerdings ist es nicht Ziel von RSerPool sämtliche Pools des Internets zu verwalten. RSerPool soll auf Bereiche wie eine Firma oder Institution beschränkt sein; allerdings heißt dies nicht, daß Server in Pools nicht weltweit im Internet verteilt sein können. Dies ist sogar sehr sinnvoll, um zu vermeiden daß ein Dienst durch lokale Katastrophen (z.B. Erdbeben oder Überschwemmung) ausfällt.

Erweiterbarkeit: Es muß möglich sein, die RSerPool-Architektur an zukünftige, neue Anwendungen anzupassen. Dazu gehört insbesondere auch die Möglichkeit, neue Auswahlprozeduren für Server hinzuzufügen. Das heißt, eine neue Applikation kann Regeln definieren, welcher Server des Pools der für einen Nutzer geeignetste ist (z.B. derjenige mit der augenblicklich niedrigsten Auslastung). Die RSerPool-Architektur muß es daher erlauben, solche Regeln zu definieren.

Einfachheit: Es soll mit möglichst wenig Aufwand verbunden sein, Server zu einem Pool hinzuzufügen bzw. wieder daraus zu entfernen. Das heißt, die Konfiguration des Systems sollte möglichst automatisch erfolgen, so daß im Regelfall nur der entsprechende Server ein- oder ausgeschaltet werden muß und sämtliche Konfiguration geschieht vollautomatisch.

Das RSerPool-Konzept

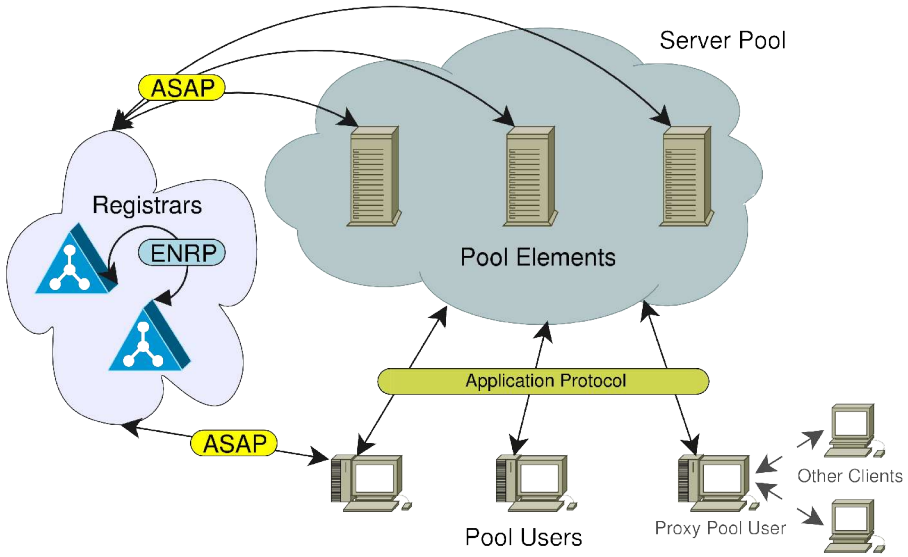


Abbildung 2: Die RSerPool-Architektur

Abbildung 2 stellt den Grundaufbau der von der RSerPool WG entwickelten RSerPool-Architektur, welche in [22] definiert wird, dar. In der Terminologie von RSerPool werden Server als Pool-Elemente (PE) bezeichnet, innerhalb ihres Pools besitzen sie eine eindeutige Kennung – ihren sogenannten PE-Identifier (PE-ID) – in Form einer zufällig gewählten 32-Bit-Nummer. Jeder Pool eines Gültigkeitsbereiches, Operation Scope genannt, ist durch einen eindeutigen Pool-Handle (PH) identifiziert. Ein PH wird durch einen beliebigen Bytevektor dargestellt, im Regelfall wird dies jedoch ein ASCII- oder Unicode-String wie z.B. „DownloadPool“ sein. Die Menge aller Pools eines Operation Scope wird Handlespace genannt. Der Aufbau des Handlespace ist „flach“, d.h. im Gegensatz zum DNS erfolgt keine hierarchische Untergliederung; er besteht einfach nur aus den durch PH identifizierten Pools.

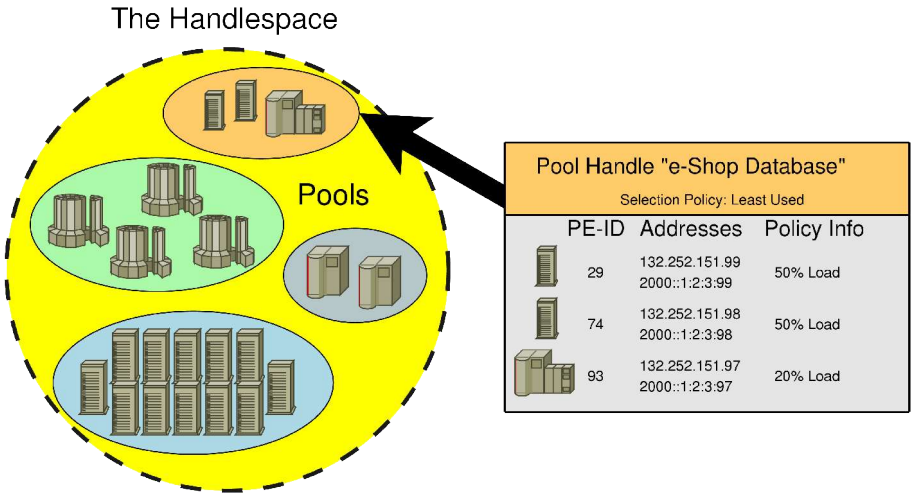


Abbildung 3: Der Handlespace

Abbildung 3 zeigt den Aufbau eines Handlespaces: Im dargestellten Beispiel

besteht der Handlespace aus vier Pools, wobei die Informationen zum ersten Pool im Kasten rechts dargestellt sind. Der gezeigte Pool ist durch seinen PH „e-Shop Database“ identifiziert, er enthält drei PEs mit ihren IDs, IPv4- sowie IPv6-Transportadressen und sogenannten Policy-Informationen zu jedem PE. Policys (Regeln zur Serverauswahl) werden im weiteren Verlauf genauer erklärt.

Poolverwaltung

Der Handlespace eines Operation Scope wird von speziellen Verwaltungskomponenten, den sogenannten Registrars (PR) verwaltet. Um zu verhindern daß ein PR als Single Point of Failure das gesamte System bei Ausfall unbrauchbar macht, sind die PRs ebenfalls redundant ausgelegt. Das heißt, in jedem Operation Scope sollte es mindestens zwei PRs geben. Die im Operation Scope vorhandenen PRs gleichen ihre Sicht auf den Handlespace mittels des Endpoint haNdlespace Redundancy Protocol (ENRP [24][18]) ab. Dies bedeutet, daß bei Ausfall eines PRs jeder andere PR des Operation Scope gleichwertig die Aufgaben des ausgefallenen übernehmen kann. PRs werden im Operation Scope anhand einer Identifikationsnummer, der Registrar-ID (PR-ID), identifiziert.

Ein PE kann sich zu einem Pool hinzufügen, indem es über das Aggregate Server Access Protocol (ASAP [17][18]) mit einem beliebigen PR des Operation Scope Kontakt aufnimmt und sich zu einem Pool registriert. Der vom PE zur Registrierung gewählte PR wird für das PE zum sogenannten Home-PR (PR-H). Er informiert nicht nur die anderen PRs des Operation Scope über Hinzufügung und Entfernung seines PEs (via ENRP), sondern überwacht diese auch durch regelmäßige Keep-Alive-Nachrichten über die ASAP-Verbindung zwischen PE und PR. Diese Keep-Alive-Nachrichten müssen vom PE innerhalb einer vorgegebenen Zeitspanne beantwortet werden. Bleibt eine Antwort aus, wird dies als Ausfall interpretiert und das PE sofort aus dem Handlespace entfernt. Zudem muß sich ein PE regelmäßig re-registrieren. Bei einer Reregistrierung hat es zudem die Möglichkeit, Registrierungsinformationen wie die Liste der Transportadressen über die es erreichbar ist oder Policy-Informationen (dies wird später erklärt werden) zu ändern. Durch die Überwachung der PEs durch ihren jeweiligen Home-PR wird erreicht, daß die im Handlespace verzeichneten PEs mit sehr hoher Wahrscheinlichkeit auch tatsächlich erreichbar sind.

Poolnutzung

Clients werden in der Terminologie von RSerPool als Pool-User (PU) bezeichnet. Um den von einem Pool angebotenen Dienst zu nutzen, fragt ein PU zuerst über das ASAP-Protokoll bei einem beliebigen PR des Operation Scope um Auflösung des PH in eine Liste von PE-Identitäten des durch den PH gegebenen Pools nach. Diese Prozedur wird Handle Resolution genannt. Der PR wird, vorausgesetzt der gewünschte Pool existiert im Handlespace, nach einer für den Pool festgelegten Auswahlregel – der sogenannten Pool Member Selection Policy oder kurz Pool Policy – eine Liste von PE-Identitäten des Pool auswählen und dem PU zurückliefern.

Zur Auswahl der PEs sind zahlreiche Verfahren denkbar, z.B. die zufällige Auswahl (Random), die Auswahl reihum (Round Robin) oder die PEs mit der augenblicklich geringsten Auslastung (Least Used). Während die ersten beiden Verfahren keine Zusatzinformationen benötigten (nicht-adaptive Verfahren), benötigt Least Used als adaptives Verfahren möglichst aktuelle Zustandsinformationen über die augenblickliche Auslastung der PEs. Diese als Policy-Informationen bezeichneten Zustände gibt ein PE bei Registrierung an; ein Aktualisierung ist per Reregistrierung möglich. Je nach Häufigkeit von Änderungen erzeugt dies natürlich entsprechenden Overhead; allerdings kann dabei natürlich auch je nach Diensteigenschaften eine deutlich bessere Lastbalancierung erreicht werden.

Nachdem der PU nun eine Liste von durch den PR ausgewählten PE-Identitäten erhalten hat, schreibt er diese in seinen lokalen Auswahlcache. Aus diesem wählt er dann – wiederum anhand der Pool Policy – genau ein PE aus, um zu diesem dann eine Verbindung aufzubauen und den Dienst des PEs mit dem Applikationsprotokoll zu nutzen. Ist eine erneute Auswahl notwendig – z.B. weil das gewählte PE nicht erreichbar war oder ausgefallen ist – kann für eine erneute Auswahl ggf. der lokale Cache genutzt werden. Dies erspart die Kommunikation mit einem PR.

Fällt ein PE während der Dienstnutzung aus, so muß nach Verbindungsaufbau zu einem neu ausgewählten PE der Sitzungszustand wiederhergestellt werden, um die Dienstnutzung fortzusetzen. Im Fall eines FTP-Datentransfers könnte dies darin bestehen, dem neuen FTP-Server den Dateinamen und die letzte empfangene Dateiposition mitzuteilen. Die Wiederherstellung einer unterbrochenen Sitzung – Failover-Prozedur genannt - ist sehr applikationsspezifisch und daher nicht direkter Teil von RSerPool. Allerdings bietet RSerPool eine umfangreiche Unterstützungsmaßnahme für eine solche Sitzungswiederaufnahme an: den Session-Layer.

Der Session-Layer

Im einfachsten Fall übernimmt RSerPool nur die Poolverwaltung (PE-Seite) und Auflösung eines PH zu einer Transportadresse (PU-Seite). In diesem Fall obliegt die Kontrolle der Transportverbindung für das Applikationsprotokoll zwischen PU und PE - der sogenannte Data Channel - vollständig der Applikation selbst.

RSerPool bietet jedoch noch mehr: einen vollständigen Session-Layer zwischen Transportschicht und Applikationsschicht. In diesem Fall baut die Applikationsschicht des PU nur eine logische Verbindung, die Session, zwischen sich selbst und einem Pool auf. Der Session-Layer, basierend auf ASAP, kümmert sich transparent für die Applikationsschicht um PE-Auswahl, Aufbau einer Verbindung, Überwachung der Verbindung, Neuauswahl eines PEs im Fehlerfall und Anstoßen einer applikationsspezifischen Failover-Prozedur zur Sitzungswiederherstellung auf einem neuen PE. Über die vom Session-Layer aufgebaute Transportverbindung zwischen PU und PE werden dabei sowohl das Applikationsprotokoll – der Data Channel - als auch eine ASAP-Kontrollverbindung – der sogenannte Control Channel – gemultiplext.

Die Failover-Prozedur ist wie bereits erwähnt applikationsspezifisch und daher nicht Teil von RSerPool selbst. Bei einem Failover kann daher ggf. die Applikationsschicht informiert werden, um eine eigene Prozedur zum Failover auszuführen. RSerPool bietet jedoch über den Control Channel ein Unterstützungsverfahren für den Failover, das in vielen Fällen schon ausreicht um einen Failover ohne direkte Unterstützung durch die PU-seitige Applikation selbst durchzuführen: Client-basiertes State-Sharing [5].

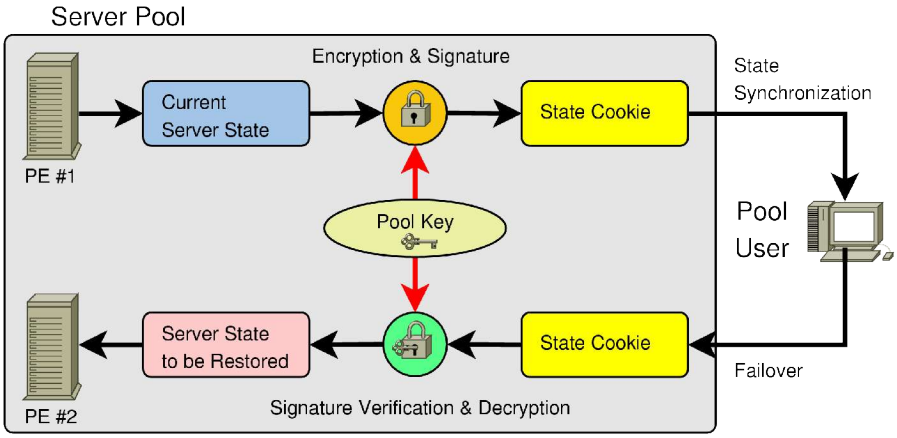


Abbildung 4: Client-basiertes State-Sharing

Die Grundidee dieses Verfahrens ist in Abbildung 4 dargestellt: Der Server verpackt die vollständigen Informationen über seinen aktuellen Sitzungszustand in ein sogenanntes State-Cookie und schickt dieses an den Client. Der Client sichert nun jeweils das zuletzt empfangene State-Cookie. Fällt der Server aus, so wird das gespeicherte State-Cookie an den neuen Server verschickt. Dieser kann nun anhand der enthaltenen Informationen den Sitzungszustand wiederherstellen. Damit der Client das State-Cookie nicht manipulieren kann, wird es vom Server im einfachsten Fall einfach mit einem allen Servern des Pools bekannten Schlüssel signiert. Der empfangende Server kann diese Signatur überprüfen und ggf. die Sitzungswiederherstellung verweigern. Zudem können Informationen im State-Cookie verschlüsselt werden, um ein Auslesen durch den Client zu verhindern.

Da das Verfahren des Client-basierten State-Sharings sehr einfach und für viele Fälle anwendbar ist, wurde es in ASAP integriert. Über den Control Channel kann ein PE Cookie-Nachrichten an den PU schicken; der Session-Layer auf PU-Seite speichert das jeweils letzte State-Cookie und schickt es bei einem Failover in Form einer Cookie-Echo-Nachricht zum neuen PE. Damit ist der Failover auf PU-Seite vollkommen transparent für die Applikationsebene. Auf PE-Seite muß die Applikationsschicht lediglich das State-Cookie auslesen und den Sitzungszustand wiederherstellen.

Der RSerPool Protokoll-Stack

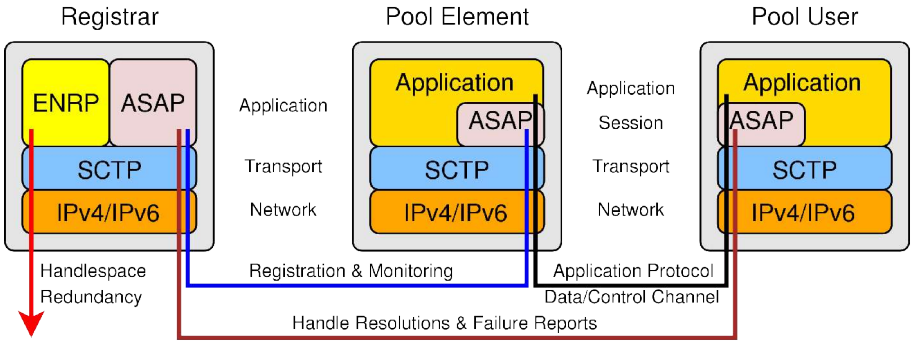


Abbildung 5: Der RSerPool-Protokollstack

Abbildung 5 zeigt den Protokollstack von RSerPool für seine drei Elementtypen: PR, PE und PU. Für den PR stellen ASAP und ENRP Applikationsprotokolle dar: Über ASAP wird für PEs die Poolverwaltungsfunktionalität angeboten, für PUs die Handle Resolution; über ENRP gleichen die PRs ihre Sicht auf den Handlespace ab. Für PE und PU stellt ASAP den im vorherigen Abschnitt beschriebenen Session-Layer dar.

Grundlegendes Transportprotokoll für ASAP und ENRP ist SCTP, aufgrund seiner hohen Toleranz gegenüber Netzwerkausfällen und insbesondere auch der Möglichkeit zum Multihoming. Für den gemultiplexten Control/Data Channel zwischen PU und PE sowie die Kommunikation zwischen PU und PR besteht jedoch auch die Möglichkeit, mittels einer Anpassungsschicht TCP zu nutzen. Allerdings ist dieser Spezialfall mit Einschränkungen - insbesondere fehlendem Multihoming - verbunden und daher nur dann sinnvoll, wenn auf dem verwendeten System (z.B. einem Embedded Device) kein SCTP-Stack verfügbar gemacht werden kann.

Automatische Konfiguration

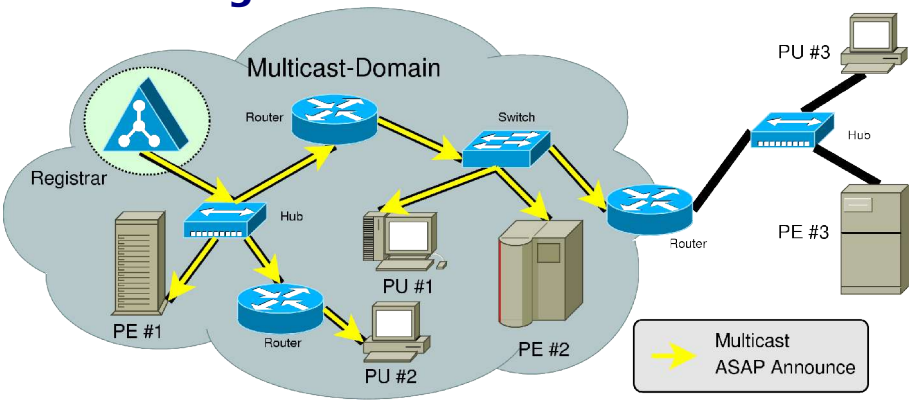


Abbildung 6: Automatische Konfiguration durch Announces

Damit ein PE oder PU die Funktionalitäten von RSerPool nutzen kann, muß im Wesentlichen nur ein PR des Operation Scope bekannt sein. Möglich ist natürlich, eine Liste von PRs statisch vorzukonfigurieren. Dies macht jedoch eine Neukonfiguration notwendig, sobald Änderungen an der Anzahl oder den Adressen von PRs gemacht werden.

Um den Aufwand einer manuellen Neukonfiguration von PR-Adressen zu vermeiden, besteht die Möglichkeit daß PRs in regelmäßigen Intervallen über

ASAP via UDP mittels IP-Multicast sogenannte Announces verschicken. Diese enthalten einfach die Information unter welchen Adressen der jeweilige PR erreichbar ist. Durch Horchen auf einer festgelegten Multicastadresse können PUs und PEs nun einfach die Liste aktuell verfügbarer PRs lernen. Die Verwendung von Multicast gegenüber Broadcast hat zudem den Vorteil, daß in einem geswitchten Ethernet nur die Stationen, die auch wirklich an den Announces interessiert sind, diese erhalten und zudem funktioniert das Verfahren auch über Routergrenzen hinweg.

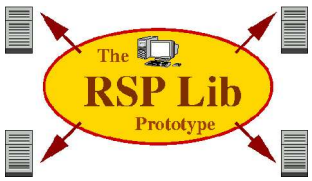
Abbildung 6 verdeutlicht noch einmal das Prinzip der Announces: Alle in der Multicast-Domäne vorhandenen PUs und PEs erfahren über Announces automatisch vom Vorhandensein des PRs; PU #3 und PE #3 befinden sich allerdings außerhalb (z.B. Rechner im Internet), daher müssen diese Elemente statisch konfiguriert werden.

Damit PRs eines Operation Scope sich gegenseitig finden, wird das gleiche Prinzip – Announces via IP-Multicast - ebenfalls für ENRP verwendet. Announces heißen hier allerdings Peer Presences.

Unsere Implementation: Der *rsplib*-Prototyp

Die Geschichte des rsplib-Projektes

Im Jahr 2002 haben wir – die Arbeitsgruppe „Technik der Rechnernetze“ des Instituts für Experimentelle Mathematik der Universität Duisburg-Essen (IEM-TdR) - damit begonnen, RSerPool als Teil eines Forschungsprojektes in Zusammenarbeit mit Siemens ICN in München und unterstützt durch das BMBF, als Prototyp zu implementieren. Ziel unseres Projektes – *rsplib Prototype* genannt – war dabei insbesondere auch, eine lauffähige Referenzimplementation des in Form mehrerer Internet-Drafts vorliegenden Konzeptes zu realisieren, um die Standardisierung von RSerPool voranzubringen.



Unsere Implementation ist Open Source und steht unter der GPL-Lizenz; siehe dazu auch unsere Projektseite [15]. Von Beginn an wurden als Plattformen GNU/Linux, FreeBSD und Darwin (MacOS X) unterstützt. Dabei wurde großer Wert darauf gelegt, die Implementation möglichst plattformunabhängig zu gestalten um so eine zukünftige Portierung auf neue Plattformen – insbesondere auch auf Geräte wie Embedded Devices – so einfach wie möglich zu gestalten. Aufgrund der Forderung nach Plattformunabhängigkeit wurde als Implementationssprache ANSI-C gewählt, da dies die am weitesten verbreitete Programmiersprache ist und auf fast jeder Plattform zu finden ist. Momentan wird an Portierungen für Solaris und M$-Windows gearbeitet, so daß mittelfristig alle wichtigen Betriebssysteme von unserem Prototyp unterstützt sein werden.

Zu Beginn unseres Projektes stand noch keine stabile Kernel-SCTP-Implementation zur Verfügung, daher wurde zuerst unsere eigene Userland-SCTP-Implementation *sctplib* [25] - ebenfalls ein Open-Source-Projekt, unter der

LGPL-Lizenz – verwendet. Mittlerweile haben sowohl GNU/Linux mit *lksctp* als auch FreeBSD mit dem KAME-Stack stabile SCTP-Implementierungen direkt im Kernel. Daher unterstützt unser Prototyp seit einiger Zeit auch diese; aufgrund der API-Kompatibilität (Socket-API analog zu TCP und UDP) aller SCTP-Implementationen ist dies relativ einfach möglich.

Die Komponenten des *rsplib*-Prototyps

Unsere aktuelle Entwicklerversion 0.8.0, zu finden unserer Webseite unter [15], enthält momentan folgendes:

- einen Registrar mit komplettem ENRP-Protokoll und Unterstützung aller in [21] definierten Policys,
- die Library *rsplib* zur Erstellung von PUs und PEs mit Unterstützung für Basic Mode API und Enhanced Mode API (Erklärung folgt unten) und
- eine Reihe von Demoprogrammen.

Die beiden Hauptkomponenten, Registrar und *rsplib*-Library werden im Folgenden nun beschrieben.

Der Registrar

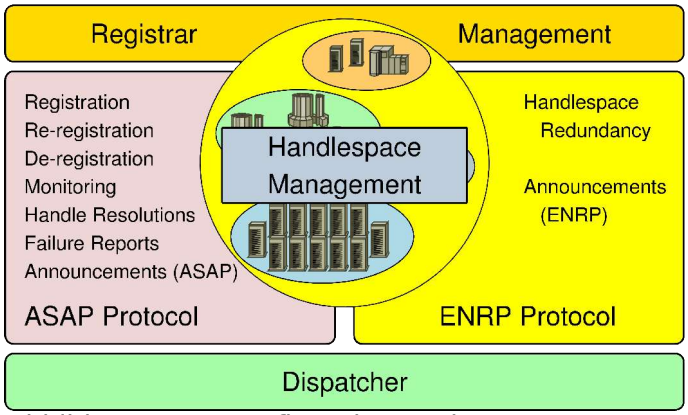


Abbildung 7: Der Aufbau des Registrars

Der grundsätzliche Aufbau unseres PRs ist in Abbildung 7 dargestellt. Wesentlicher Bestandteil des PRs ist – neben der Implementation der Protokolle ENRP und ASAP – die Verwaltung des Handlespaces. In die Entwicklung der notwendigen Verwaltungsstrukturen wurde ein Hauptteil der Entwicklungsarbeit für den PR investiert: Die naive Lösung zur Verwaltung des Handlespaces besteht darin, diesen einfach aus einer Liste von Pools bestehen zu lassen und in jedem Pool eine Liste seiner PEs zu halten. Auf diese Art und Weise wurde Version 0.1.0 realisiert. Allerdings bringt dies erhebliche Skalierbarkeitsprobleme mit sich. Eine unserer Ideen ist es, RSerPool für die Verwaltung von Distributed-Computing-Pools zu nutzen. In diesem Fall können Pools natürlich hunderte oder gar tausende von Rechnern enthalten. Zudem

muß je nach Anwendung ggf. eine eigene Auswahlpolicy definiert werden können. Die Größe von Pools sowie die Vielzahl möglicher Auswahlregeln lassen die Verwaltung durch Listen sehr schnell ineffektiv werden. Um den Handlespace effizient zu verwalten haben wir dazu auf Red-Black-Trees (dies sind balancierte Binärbäume) basierende Datenstrukturen und Verwaltungsalgorithmen entwickelt. Eine ausführlichere Behandlung der Strukturen würde den Rahmen dieses Artikels sprengen; zu Details siehe daher unsere Veröffentlichung [10].

Der in der Abbildung als Dispatcher dargestellte Block stellt die systemabhängige Kapselung von Timer-, Thread- und Socketfunktionalitäten dar. Wie bereits bei den Grundideen zu unserem Prototyp beschrieben, soll unser System möglichst leicht portierbar sein. Soweit möglich wurden daher sämtliche betriebssystemabhängigen Teile im Dispatcher gekapselt, so daß bei Portierung auf ein neues System nur dessen Bestandteile entsprechend modifiziert werden müssen.

Die *rsplib*-Library

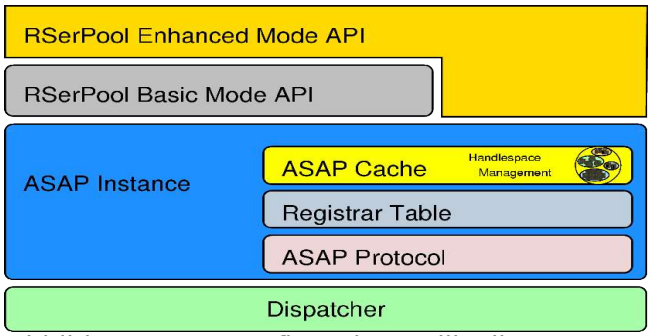


Abbildung 8: Der Aufbau der rsplib-Library

Die *rsplib*-Library stellt für Server und Clients die Schnittstelle zu RSerPool dar, d.h. sie enthält die Funktionen zur Implementation von PEs und PUs. Im Wesentlichen besteht diese Library aus einer auf dem Dispatcher zur Kapselung plattformabhängigen Codes aufbauenden Implementation einer ASAP-Instanz, ihr Aufbau ist in Abbildung 8 dargestellt. Die ASAP-Instanz besteht aus drei Blöcken:

ASAP Protocol: Dies ist die Implementation des ASAP-Protokolls.

Registrar Table: In der Registrar Table wird eine aktuelle Liste von verfügbaren PRs vorgehalten. Die Einträge dieser Liste werden dabei dynamisch durch Multicast-Announces der entsprechenden PRs gelernt (automatische Konfiguration) oder statisch vorgegeben.

ASAP Cache: Dieser Cache sorgt beim PU für eine Zwischenspeicherung der vom PR nach Handle Resolution erhaltenen PE-Liste. Vom Prinzip her ist der Cache ein Ausschnitt des kompletten Handlespaces; die Algorithmen und Datenstrukturen zu dessen Verwaltung – von uns für den Registrar entwickelt – lassen sich hier einfach wiederverwenden.

Aufbauend auf der ASAP-Instanz ist das eigentliche API der *rsplib*-Library realisiert. Dieses API besteht aus zwei wesentlichen Teilen: dem Basic Mode API und dem darauf aufbauenden Enhanced Mode API. Mit dem Basic Mode stellt die *rsplib*-Library nur die wesentlichen ASAP-Funktionen zur Poolverwaltung und -nutzung zur Verfügung. Für PEs wären das die Registrierung, Reregistrierung und Deregistrierung; für PUs gibt es die Möglichkeit zur Handle Resolution. Die komplette Verwaltung des Data Channels obliegt dabei dann der Anwendung selbst.

Erst mit Verwendung des Enhanced Mode APIs wird RSerPool zu einem wirklichen Session-Layer; in diesem Fall übernimmt die *rsplib*-Library PE-Auswahl, Verbindungsaufbau, Multiplexing von Control und Data Channel, Ausfallerkennung und Failover-Unterstützung. Für die Implementation des Enhanced Mode APIs wird dabei im Wesentlichen das Basic Mode API verwendet um den Session-Layer zu realisieren.

Im folgenden Kapitel wird ein Überblick über beide Modi des *rsplib*-APIs einschließlich Code-Beispielen gegeben.

Die Benutzung der *rsplib*-Library

In diesem Abschnitt geben wir einen kurzen Überblick über das Basic- und Enhanced Mode API der *rsplib*-Library.

Das Basic Mode API

```
void registrationLoop()
{
   struct timeval timeout;
   ...
   while(!shuttingDown) {
      rspRegister(poolHandle, ...);
      timeout.tv_sec  = reregistrationInterval;
      timeout.tv_usec = 0;
      rspSelect(0, NULL, NULL, NULL, &timeout);
   }
   rspDeregister(poolHandle, ...);
}
```

Algorithmus 1: Pseudocode-Beispiel für einen PU mit dem Basic Mode API

Die Handhabung der PE-Funktionalität mit dem Basic Mode API ist relativ offensichtlich: Die Server-Applikation muß zur Registrierung bzw. Reregistrierung (inkl. Update der Policy-Informationen) nur eine Registrierungsfunktion aufrufen, welcher alle benötigten Parameter (PH, Transportadressen, Policy-Informationen) übergeben werden. Das heißt, es genügt in regelmäßigen Abständen bzw. bei Notwendigkeit der Aktualisierung von Informationen die Funktion *rspRegister*() mit den entsprechenden Parametern aufzurufen. Üblicherweise wird eine solche Funktionalität in einem separaten Thread erledigt, so daß Verzögerungen – z.B. die Suche eines PRs –

den Dienst des Servers nicht beeinflussen.

Ein Beispiel für eine solche Thread-Funktion ist in Algorithmus 1 gezeigt: Solange der Server aktiv ist, wird in regelmäßigen Abständen (gegeben durch *reregistrationInterval* in Sekunden) die (Re-)Registrierung durchgeführt. Die Funktion *rspSelect*() wartet die entsprechende Zeitspanne (analog zum *select* ()-Aufruf) und beantwortet zudem ASAP Keep-Alives des Home-PR. Möglich wäre hier zudem – wie bei *select*() - das Warten bei Socket-Ereignissen zu deren Behandlung zu unterbrechen.

Zur Deregistrierung aus dem Pool wird schließlich *rspDeregister*() aufgerufen.

```
poolHandle = "DownloadPool";
rspHandleResolution(poolHandle, strlen(poolHandle), &eai);
...
sd = socket(eai->ai_family, eai->ai_socktype, eai ->ai_protocol);
if(sd >= 0) {
   if(connect(sd, eai->ai_addr, eai->ai_addrlen) {
      ...
      if(failure) {
         rspReportFailure(poolHandle, strlen(poolHandle),
                          eai->ai_identifier);
         ...
      }
      ...
   }
}
```

Algorithmus 2: Pseudocode-Beispiel für einen PU mit dem Basic Mode API

Ein Beispiel für die PU-seitige Nutzung von RSerPool mit dem Basic Mode API ist in Algorithmus 2 dargestellt: Statt den Hostnamen des Servers durch DNS in eine IP-Adresse auflösen zu lassen, wird die Funktion *rspHandleResolution*() zur Auflösung eines PH (hier: "DownloadPool") in eine durch Pool Policy gewählte PE-Identität aufgerufen. Diese Funktion übernimmt dabei die Kommunikation mit PR, Cache-Verwaltung und Serverauswahl. Ihre Parameter und Strukturen sind dabei kompatibel zum DNS-Aufruf *getaddrinfo*(), so daß die zur RSerPool-Nutzung notwendigen Änderungen am Programm minimal sind. Nach der Auswahl eines PEs wird wie bei einer herkömmlichen Client-Anwendung ein Socket für das entsprechende Netzwerkprotokoll (i.d.R. IPv4 oder IPv6) und Transportprotokoll (z.B. SCTP oder TCP) erzeugt (*socket*()-Aufruf) und eine Verbindung zur Dienstnutzung hergestellt (*connect*()-Aufruf). Im Fehlerfall muß sich der PU selbst um Neuauswahl eines Servers (wieder durch Aufruf von *rspHandleResolution*()) und Failover kümmern.

Eine etwas ausführlichere Beschreibung unseres Basic Mode APIs ist in [12] zu finden.

Enhanced Mode API

```
sd = rspCreateSession("DownloadPool", ...);
rspSessionWrite(sd, "GET Linux-CD.iso HTTP/1.0\r\n\r\n");
while((length = rspSessionRead(sd, buffer, ...)) > 0) {
   doSomething(buffer, length);
}
rspSessionClose(sd);
```

Algorithmus 3: Pseudocode-Beispiel für einen PU mit dem Enhanced Mode API

Mit dem Enhanced Mode API wird die Session-Layer-Funktionalität von RSerPool nutzbar. Grundidee des APIs ist es, Sessions ähnlich wie TCP-Sockets zu realisieren, so daß die Portierung bestehender Anwendungen so einfach wie möglich ist. Ein Beispiel für die PU-Programmierung ist dazu in Algorithmus 3 dargestellt: Der PU stellt eine logische Verbindung mit einem durch PH gegebenen Pool her (*rspCreateSession*()-Aufruf mit PH "DownloadPool"). Die eigentliche Verbindungsaufnahme mit einem PE des Pools erfolgt dabei für die Applikation vollkommen transparent im Session-Layer. In der Applikation selbst kann nun auf der logischen Verbindung das eigentliche Applikationsprotokoll genutzt werden. Im Beispiel ist dies HTTP, es wird die Datei "Linux-CD.iso" per GET-Befehl angefordert und anschließend übertragen. Der ASAP Session-Layer sorgt dabei für die Zuordnung der Kommunikation zu einem PE des Pools.

Einen Failover mit State-Cookie kann der Session-Layer vollautomatisch handhaben. Senden die PEs aus dem Beispiel ihren Zustand (Dateiname und aktuelle Übertragungsposition) laufend per ASAP Cookie, so kann die Übertragung durch ein neues PE automatisch vom Session-Layer wiederaufgenommen werden. Im Applikationscode des PUs selbst ist dafür nichts weiter zu tun.

```
void serviceThread(session)
{
   rspSessionRead(session, command, ...);
   if(command is cookie) {
      Restore state;
      rspSessionRead(session, command, ...);
   }
   do {
      Handle command;
      rspSessionSendCookie(session, Current state);
      rspSessionRead(session, command, ...);
   } while(Session is alive);
}

int main(...)
{
   poolElement = rspCreatePoolElement(poolHandle, ...);
   while(!shuttingDown) {
      rspSessionSelect(... poolElement ...);
      if(New Session) {
         session = rspAcceptSession(poolElement, ...);
         Create service thread for new session;
      }
   }
   rspDeletePoolElement(poolElement);
}
```

Algorithmus 4: Pseudocode-Beispiel für ein PE mit dem Enhanced Mode API

Die Struktur der Realisierung eines Thread-basierten PEs mit dem Enhanced Mode API ist in Algorithmus 4 dargestellt: In der *main*()-Funktion wird ein PE mit *rspCreatePoolElement*() erzeugt; diese Funktion sorgt auch für die Registrierung. In einem Hintergrundthread der *rsplib*-Library wird danach zudem automatisch für Reregistrierung und Beantwortung der ASAP Keep-Alives gesorgt. Analog zu Sockets wird nach Registrierung des PEs auf eingehende Sessions gewartet (*rspSessionSelect*(), analog zum *select*()-Aufruf für Sockets). Wurde eine neue Session aufgebaut, nimmt der Server diese mit *rspAcceptSession*() (analog zum *accept*()-Aufruf für Sockets) an und erzeugt einen neuen Thread für ihre Bedienung (in *serviceThread*()).

Erste Aufgabe des Service-Threads ist es festzustellen, ob eine Sitzungswiederherstellung durchgeführt werden muß; hier: basierend auf dem in RSerPool eingebauten Mechanismus des Client-basierten State-Sharings mit State-Cookies. Wurde ein State-Cookie vom PU geschickt (in Form eines ASAP Cookie-Echo über den Control Channel), wird der in ihm gespeicherte Zustand wiederhergestellt (sofern dieser gültig ist!).

In der while-Schleife werden nun die Kommandos des PUs abgearbeitet. Nach Ausführung jedes Kommandos wird ein Cookie mit dem aktuellen Zustand der Sitzung geschickt. Das heißt, die Sitzung könnte dann an dem durch das State-Cookie beschriebenen Punkt fortgesetzt werden.

Unser Demosystem

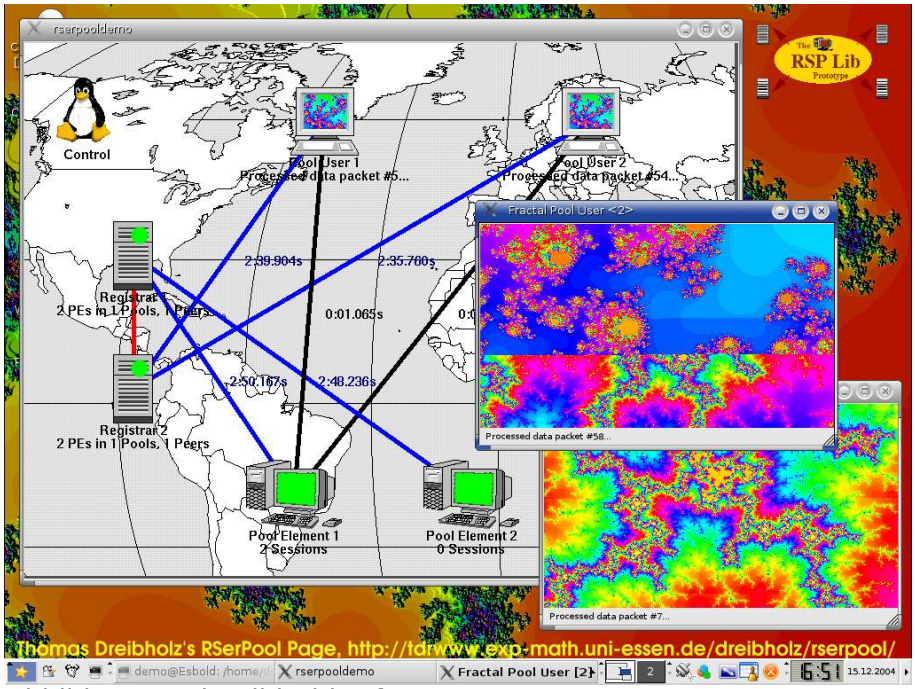


Abbildung 9: Ein Bildschirmfoto unseres Demosystems

Um die Arbeit unseres Prototyps anschaulich demonstrieren zu können, haben wir basierend auf dem Qt-Toolkit ein kleines Demosystem [9] entwickelt. Ein Bildschirmfoto davon ist in Abbildung 9 zu sehen.

Bei unserem System handelt es sich um ein Beispiel zur Anwendung von RSerPool im Bereich des Distributed Computung: Ein Pool besteht aus PEs, welche einen Dienst zur Berechnung von Mandelbrot'schen Fraktalgrafiken anbieten. Das Berechungsverfahren dazu ist sehr bekannt, relativ einfach und die Ergebnisse recht anschaulich. PUs nutzen diesen Berechnungsdienst und stellen den Rechenfortschritt kontinuierlich auf dem Bildschirm dar. Das heißt: sobald der Service ausfällt wird dies für den Betrachter sofort sichtbar.

Mittels einer Kontrollkomponente ist das Starten und Stoppen einzelner Komponenten möglich. Diese Kontrollkomponente zeigt zudem laufend den aktuellen Status der Verbindungen (ASAP, ENRP, Applikation) zwischen den einzelnen Einheiten des Szenarios an. Die Funktionalität der Kontrollkomponente basiert im Wesentlichen auf der Ausführung von *bash*-Skripten. Damit ist es in Verbindung mit *ssh* sogar möglich die einzelnen Komponenten auf mehreren Rechnern laufen zu lassen. Werden nun Komponenten abgeschaltet – per GUI oder indem einfach der Netzwerkstecker gezogen wird – so werden die Auswirkungen auf das Verhalten des Systems unmittelbar sichtbar: die RSerPool-Mechanismen sorgen für einen Failover zu einem neuen PE des Pools.

PU und PE basieren auf dem Enhanced Mode API der *rsplib*-Library. Das heißt, in der Implementierung der Komponenten selbst ist der Aufwand für RSerPool relativ gering. Für den Failover werden State-Cookies verwendet – das PE

speichert dort seine aktuelle Rechenposition sowie sämtliche Parameter für den Mandelbrot-Algorithmus – so daß auf der PU-Seite in der Applikationsschicht überhaupt nichts für die Failover-Funktionalität zu tun ist: Die Applikationsschicht des PU baut eine Session zum Berechnungspool auf und startet die Berechnung – der PE-Wechsel bei Ausfall ist für sie vollkommen transparent. Die Applikationsschicht des PEs muß zur Failover-Unterstützung lediglich den Sitzungszustand aus dem State-Cookie übernehmen.

Bei unserer Präsentation werden wir selbstverständlich das Demosystem einmal vorführen.

Unsere Aktivitäten in der Standardisierung

Wie bereits in der Einführung zu unserem Prototypen erwähnt, wurde dieser auch dazu entwickelt, die Standardisierung von RSerPool innerhalb der IETF voranzubringen. Dieses Ziel konnte erreicht werden: Durch Erfahrungen aus der Implementierung der vorher nur als Internet-Drafts vorliegenden Protokolldefinitionen konnten einige Fehler und Schwachstellen in diesen Dokumenten gefunden und korrigiert werden. Zudem haben wir als Beitrag einen eigenen Draft zu Pool Policys erstellt [21], welcher mittlerweile von der RSerPool WG zum Working-Group-Dokument erhoben wurde.

Unser Prototyp ist zur Zeit die einzige Open Source Implementation von RSerPool; sie wird daher von der IETF RSerPool WG auch als Referenzimplementation verwendet. Neben unserem Prototyp gibt es noch eine proprietäre, nicht öffentliche Realisierung von RSerPool durch Motorola. Bei einem inoffiziellen Interoperabilitätstest auf dem 60. IETF-Meeting im August 2004 konnte erfolgreich die Zusammenarbeit beider Implementierungen getestet werden. Somit gibt es nun zwei verschiedene, interoperable Implementationen von RSerPool, womit die Standardisierungsbemühungen einen wesentlichen Schritt vorangekommen sind. Es ist nun damit zu rechnen, daß relativ bald weitere Internet-Drafts von RSerPool zu RFCs erhoben werden und damit dann anerkannter Standard sein werden.

Zusammenfassung

In diesem Artikel haben wir zunächst einen Überblick über die Vorgeschichte von Reliable Server Pooling (RSerPool) gegeben: die Anforderungen der Telekommunikationssignalisierung über IP-Netzwerke und das SCTP-Protokoll. Im Anschluß daran haben wir die RSerPool-Architektur mit ihren Funktionalitäten zur Poolverwaltung, Serverauswahl und Dienstnutzung vorgestellt.

Im Folgenden stellten wir unser Projekt *rsplib Prototype* - eine Open Source Prototypimplementierung von RSerPool – vor. Unsere Implementation besteht aus einem Registrar und einer Library für PEs und PUs. Die beiden Teile des APIs der Library, Basic Mode API und Enhanced Mode API, haben wir im Anschluß kurz beschrieben und mit Codebeispielen deren Verwendung verdeutlicht. Darauf hin folgte noch ein Überblick über unser Demo-System,

das wir bei der Präsentation auch kurz vorführen möchten. Zum Schluß haben wir noch eine Zusammenfassung unserer Aktivitäten bei der Standardisierung von RSerPool in der IETF gegeben.

Referenzen

[1] COENE, L., CONRAD, P., AND LEI, P. Reliable Server pool applicability Statement. Internet-Draft Version 01, IETF, RSerPool WG, Oct 2003. draft-ietf-rserpool-applic-01.txt.

[2] CONRAD, P., JUNGMAIER, A., ROSS, C., SIM, W.-C., AND TÜXEN, M. Reliable IP Telephony Applications with SIP using RSerPool. In *Proceedings of the SCI 2002, Mobile/Wireless Computing and Communication Systems II* (Orlando/U.S.A., Jul 2002), vol. X.

[3] CONRAD, P., AND LEI, P. Services Provided By Reliable Server Pooling. Internet-Draft Version 01, IETF, RSerPool WG, Jun 2004. draft-ietf-rserpool-service-01.txt, work in progress.

[4] CONRAD, P., AND LEI, P. TCP Mapping for Reliable Server Pooling Enhanced Mode. Internet-Draft Version 02, IETF, RSerPool WG, Jun 2004. draft-ietf-rserpool-tcpmapping-02.txt, work in progress.

[5] DREIBHOLZ, T. An efficient approach for state sharing in server pools. In *Proceedings of the 27th IEEE Local Computer Networks Conference* (Tampa, Florida/U.S.A., Oct 2002).

[6] DREIBHOLZ, T. An Overview of the Reliable Server Pooling Architecture. In *Proceedings of the 12th IEEE International Conference on Network Protocols 2004* (Berlin/Germany, Oct 2004).

[7] DREIBHOLZ, T., COENE, L., AND CONRAD, P. Reliable Server pool use in IP flow information exchange. Internet-Draft Version 01, IETF, Individual submission, Apr 2005. draft-coene-rserpool-applic-ipfix-01.txt.

[8] DREIBHOLZ, T., JUNGMAIER, A., AND TÜXEN, M. A new Scheme for IP-based Internet Mobility. In *Proceedings of the 28th IEEE Local Computer Networks Conference* (Königswinter/Germany, Nov 2003).

[9] DREIBHOLZ, T., AND RATHGEB, E. P. An Application Demonstration of the Reliable Server Pooling Framework. In *Proceedings of the 24th IEEE Infocom 2005* (Miami, Florida/U.S.A., Mar 2005).

[10] DREIBHOLZ, T., AND RATHGEB, E. P. Implementing of the Reliable Server Pooling Framework. In *Proceedings of the 8th IEEE International Conference on Telecommunications 2005* (Zagreb/Croatia, Jun 2005).

[11] DREIBHOLZ, T., RATHGEB, E. P., AND TÜXEN, M. Load Distribution Performance of the Reliable Server Pooling Framework. In *Proceedings of the 4th IEEE International Conference on Networking 2005* (Saint Gilles Les Bains/Reunion Island, Apr 2005).

[12] DREIBOLZ, T., AND TÜXEN, M. High availability using reliable server pooling. In *Proceedings of the Linux Conference Australia 2003* (Perth/Australia, Jan 2003).

[13] GRADISCHNIG, K. D., AND TÜXEN, M. Signaling transport over IP-based networks using IETF standards. In *Proceedings of the 3rd International Workshop on the design of Reliable Communication Networks* (Budapest, Hungary, 2001), pp. 168–174.

[14] RAMALHO, M., XIE, Q., TÜXEN, M., AND CONRAD, P. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. Internet-Draft Version 09, IETF, Transport Area WG, Jun 2004. draft-ietf-tsvwg-addip-sctp-09.txt, work in progress.

[15] Thomas Dreibholz's RSerPool Page.http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool.

[16] STEWART, R., XIE, Q., MORNEAULT, K., SHARP, C., SCHWARZBAUER, H., TAYLOR, T., RYTINA, I., KALLA, M., ZHANG, L., AND PAXSON, V. Stream Control Transmission Protocol. Standards Track RFC 2960, IETF, Oct 2000.

[17] STEWART, R., XIE, Q., STILLMAN, M., AND TÜXEN, M. Aggregate Server Access Protcol (ASAP). Internet-Draft Version 11, IETF, RSerPool WG, Feb 2005. draft-ietf-rserpool-asap-11.txt, work in progress.

[18] STEWART, R., XIE, Q., STILLMAN, M., AND TÜXEN, M. Aggregate Server Access Protocol (ASAP) and Endpoint Handlespace Resolution Protocol (ENRP) Parameters. Internet-Draft Version 08, IETF, RSerPool WG, Feb 2005. draft-ietf-rserpool-common-param-08.txt, work in progress.

[19] STILLMAN, M., GOPOL, R., SENGODAN, S., GUTTMAN, E., AND HOLDREGE, M. Threats Introduced by Rserpool and Requirements for Security. Internet-Draft Version 04, IETF, RSerPool WG, Jan 2005. draft-ietf-rserpool-threats-04.txt.

[20] STONE, J., STEWART, R., AND OTIS, D. Stream Control Transmission Protocol (SCTP) Checksum Change. Standards Track RFC 3309, IETF, Sep 2002.

[21] TÜXEN, M., AND DREIBHOLZ, T. Reliable Server Pooling Policies. Internet-Draft Version 00, IETF, RSerPool WG, Oct 2004. draft-ietf-rserpool-policies-00.txt, work in progress.

[22] TÜXEN, M., XIE, Q., STEWART, R., SHORE, M., LOUGHNEY, J., AND SILVERTON, A. Architecture for Reliable Server Pooling. Internet-Draft Version 09, IETF, RSerPool WG, Feb 2005. draft-ietf-rserpool-arch-09.txt, work in progress.

[23] TÜXEN, M., XIE, Q., STEWART, R., SHORE, M., ONG, L., LOUGHNEY, J., AND STILLMAN, M. Requirements for Reliable Server Pooling. Informational RFC 3237, IETF, Jan 2002.

[24] XIE, Q., STEWART, R., STILLMAN, M., TÜXEN, M., AND SILVERTON, A. Endpoint Name Resolution Protcol (ENRP). Internet-Draft Version 11, IETF, RSerPool WG, Feb 2005. draft-ietf-rserpool-enrp-11.txt, work in progress.

[25] sctplib: SCTP Prototype Implementation, http://www.sctp.de/sctp.html