Self-Adaptive Discovery Mechanisms
for Improved Performance in Fault-Tolerant Networks

Kevin Mills, Doug Montgomery, Scott Rose,
Stephen Quirolgico, and Chris Dabrowski
NIST

Mackenzie Britton (SMU) and Kevin Bowers (RPI & CMU)
NSF Summer University Research Fellows

DARPA FTN PI Meeting
July 22, 2003

Self-Organizing Systems for
Hostile & Volatile Environments
Presentation Outline

- One-Page Review of Project Objective and Plan

- One-Page Refresher on Service-Discovery Protocols

- Summary of Accomplishments on the Project (to date)

- An Autonomic Failure-Detection Algorithm for Distributed Systems
  - Applied to Service Registration in the Service Location Protocol (SLP)
  - Applied to Jini Leasing *(SEE DEMO on WEDNESDAY EVENING)*

- Performance of Service-Discovery Systems under Node Failure

- Plan for Final Six Months

- Conclusions
Project Objective


Project Plan – Three Tasks

- **TASK I** – characterize the fault-tolerance and performance of selected service discovery protocols [Universal Plug-and-Play (UPnP), Service Location Protocol (SLP) and Jini] as specified and implemented
  - develop simulation models for each protocol
  - establish performance benchmarks based on default or recommended parameter values and on required or most likely implementation of behaviors

- **TASK II** – design, simulate, and evaluate self-adaptive algorithms to improve performance of discovery protocols regarding selected mechanisms
  - devise algorithms to adjust control parameters and behavior in each protocol
  - simulate performance of each algorithm against benchmark performance
  - select most promising algorithms for further development

- **TASK III** – implement and validate the most promising algorithms in publicly available reference software
Service-Discovery Protocols in Essence

Dynamic multi-party protocols that enable distributed services:
(1) to discover each other without prior arrangement,
(2) to describe opportunities for collaboration,
(3) to compose themselves into larger collections that cooperate to meet an application need, and
(4) to detect and adapt to changes in topology.

Selected First-Generation Service-Discovery Protocols

<table>
<thead>
<tr>
<th>3-Party Design</th>
<th>2-Party Design</th>
<th>Adaptive 2/3-Party Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>JINI</td>
<td>UPnP</td>
<td>SLP</td>
</tr>
<tr>
<td>Vertically Integrated 3-Party Design</td>
<td>Network-Dependent 3-Party Design</td>
<td>Network-Dependent 2-Party Design</td>
</tr>
</tbody>
</table>
Summary of Accomplishments on the Project (as of July 2003)

TASK I

- Developed and publicly released simulation models for Jini, UPnP, and SLP
  - SLX™ discrete-event simulations for Jini, UPnP, and SLP
  - Rapide simulations for Jini and UPnP

- Characterized response of Jini and UPnP to various types of failure

- Characterized failure response in SLP for communications failure, message loss, and power-failure restart (*currently working on node-failure case*)
Summary of Accomplishments on the Project (as of July 2003)

TASK II

● Designed algorithms to automatically self-regulate performance of various service-discovery functions
  ▪ Adaptive Jitter-Control Algorithm for Multicast Search in UPnP
  ▪ Autonomic Failure-Detection Algorithm (including analysis) – MORE ON THIS AHEAD
  ▪ Self-adaptive Inverted Leasing Algorithm for Jini (including analysis)

● Developed and publicly released SLX™ discrete-event simulation models
  ▪ Adaptive Jitter Control Algorithm for UPnP M-Search
  ▪ Autonomic Leasing Algorithm for Jini – MORE ON THIS AHEAD
  ▪ Inverted Leasing Algorithm for Jini
  ▪ Autonomic Service Registration and Refresh Algorithm for SLP – MORE ON THIS AHEAD

● Published algorithms and performance characterizations
TASK III

- Implemented Autonomic Lease-Granting Algorithm for Jini
  - Modified Jini Lookup Service code (publicly released by Sun Microsystems)
  - Modified Jini Lookup-Service Administrative Interface to input policy parameters
  - Implemented test system software and infrastructure to generate, control, and monitor thousands of Jini services

- Demonstrated Autonomic Leasing for Jini
  - DISCEX III in April 2003 (Washington, D.C.)
  - Self-Managing Systems Workshop in June 2003 (San Diego, CA)
  - FTN PI Meeting in July 2003 (Honolulu, HI) – PLEASE STOP BY WEDNESDAY EVENING

- Validated Autonomic Leasing Algorithm for Jini – MORE ON THIS AHEAD
  - Deployed modified Jini Lookup Service and test system and conducted controlled experiments, collecting data for analysis
  - Implemented an analytical model of the autonomic leasing algorithm and evaluated it with the same parameters used in the live experiments
  - Iterated the Jini autonomic leasing simulation model with the same parameters used in the live experiments
  - Compared results – correspondence quite good among the measured, simulated, and analytical results
Presentation Outline

- One-Page Review of Project Objective and Plan
- One-Page Refresher on Service-Discovery Protocols
- Summary of Accomplishments on the Project (to date)

- An Autonomic Failure-Detection Algorithm for Distributed Systems
  - Applied to Service Registration in the Service Location Protocol (SLP)
  - Applied to Jini Leasing (SEE DEMO on WEDNESDAY EVENING)

- Performance of Service-Discovery Systems under Node Failure

- Plan for Final Six Months

- Conclusions
Heartbeat-Based Failure-Detection in Distributed Systems

For a given heartbeat rate and message size, the larger the patient population, the greater the bandwidth consumption.

\[ S_R = \text{size of Rising Heartbeat Message} \]
\[ S_F = \text{size of Falling Heartbeat Message} \]
An Autonomic Failure-Detection Algorithm for Distributed Systems

**Goal:** limit bandwidth usage to $B_A$ and assure avg. worst-case failure-detection latency ($L_{WORST}$), while achieving better avg. failure-detection latency $L < L_{WORST}$ when $N < N_{MAX}$

**Analysis**

\[
H_{MAX} = 2L_{WORST}
\]

Avg. worst-case failure-detection latency determines maximum heartbeat period

\[
C = \frac{B_A}{(S_R + S_F)}
\]

Allocated bandwidth $B_A$ and size of rising $S_R$ and falling $S_F$ heartbeat messages determine system capacity in heartbeats per second

\[
H_{MIN} = \frac{1}{C}
\]

Assuming minimum system size of 1, $C$ determines minimum heartbeat period

\[
H_{MIN} = 2L_{BEST}
\]

However, $1/C$ might place too great a load on an individual heart, so instead choose a avg. best-case failure-detection latency to determine minimum heartbeat period

\[
H_{MIN} \leq H_P \leq H_{MAX}
\]

Vary the heartbeat period within this range, using the following algorithm

```plaintext
set $H_P = N / C$;
if $H_P > H_{MAX}$
    then refuse to monitor the heartbeat;
elseif $H_P < H_{MIN}$
    then set $H_P = H_{MIN}$;
endif
endif
```
Analysis of Autonomic Heartbeat Algorithm in Operation

Parameters

- $L_{\text{Worst}} = 30.0$ s
- $L_{\text{Best}} = 7.5$ s
- $B_A = 576$ B/s
- $S_R = 128$ Bytes
- $S_F = 64$ Bytes

$N_{\text{Max}} = N$  
$0 < N < 150$  
$150 < N < 200$
Autonomic Heartbeat Algorithm Applied to SLP Service Registration & Refresh

**Key**
- Red Lines = Simulated
- Blue Lines = Analytical

### Parameters
- $L_{\text{WORST}} = 1200$ s
- $L_{\text{BEST}} = 7.5$ s
- $B_A = 396$ B/s
- $S_R = 76$ Bytes
- $S_F = 56$ Bytes
Autonomic Heartbeat Algorithm Applied to Jini Leasing (SEE WEDS. DEMO)

Key
Red Lines = Measured
Blue Lines = Analytical

Parameters
$L_{WORST}$ = 1200 s
$L_{BEST}$ = 7.5 s
$B_A$ = 2100 B/s
$S_R$ = 350 Bytes
$S_F$ = 350 Bytes
Presentation Outline

- One-Page Review of Project Objective and Plan
- One-Page Refresher on Service-Discovery Protocols
- Summary of Accomplishments on the Project (to date)
- An Autonomic Failure-Detection Algorithm for Distributed Systems
  - Applied to Service Registration in the Service Location Protocol (SLP)
  - Applied to Jini Leasing (SEE DEMO on WEDNESDAY EVENING)
- Performance of Service-Discovery Systems under Node Failure
- Plan for Final Six Months
- Conclusions
How Well Do Service-Discovery Protocols Replace Services Lost to Node Failure?

Mobile command post Service User (SU) dynamically locates and combines sensors and actuators.

“Fast” sensors send readings every 2 seconds.

“Slow” sensors send readings every 30 seconds.

SU is Service User
SM is Service Manager
Compared two architectures used by most service discovery protocols

- In 2-party architecture, SU discovers SMs through multicast search strategy
  - SU registers for notification of change in status of service (renewed every 300s)
- In 3-party architecture, both SMs and SUs discover SCMs; SU obtains services through SCM intermediary
  - SMs register services (renewed every 300s for fast sensors and every 60s for slow sensors and actuators); SU registers notification requests (renewed every 300s)
- SU detects failure of services through (1) non-response or (2) notification of registration renewal failure (heartbeat mechanism). Upon loss of service……
  - 2-party SU multicasts queries to SMs every 120s
  - 3-party SU queries SCMs for service; If SCMs lost, SU (and SMs) listen for SCM announcements (every 120s)
Experiment Design

Goal of SU is to be functional; i.e, to continually possess one instance of each type of service (“fast” sensor, “slow” sensor, & actuator).

- When >= 1 type of sensor is missing, SU is non-functional
- To focus on alternative architectures & associated processes, mechanisms such as service caching factored out

Q = end of quiescent period (60 s)
D = end of experiment (1800 s)
R = failure rate (variable from 0% - 80% in 10% increments)

Variations Considered (but not all results reported here):

1. After a failure period, a failed node could either be Replaced (i.e, cached data does not persist) or Restored (i.e., cached data may persist)
2. At least one SM of each type could be required to remain operational or all SMs could be allowed to fail
3. All SCMs are always allowed to fail
**Functional Effectiveness of Two-Party vs. Three-Party When One SM of Each Type is Always Available**

Measures the proportion of time the Service User possesses the operational set of remote services needed to accomplish its task during $D$.

**Notes:**

1. Replacement case
2. 60 repetitions per data points
Non-Functional Time Decomposed Proportionately into Detection Latency and Recovery Latency

Two-Party Architecture: Failure-Detection Latency dominates at all failure rates.

Three-Party Architecture (3 SCMs): Failure-Detection Latency and Recovery Latency come closer as Failure Rate increases because SCMs are required for rendezvous, but can all be failed.
Efficiency of Two-Party vs. Three-Party
When One SM of Each Type is Always Available

Notes:
(1) Replacement case
(2) 30 repetitions per data points

Measures the average number of messages required during D.

- Two Party: Slope = +2.37
- Three Party, 1 SCM: Slope = -10.55
- Three Party, 2 SCMs: Slope = -6.97
- Three Party, 3 SCMs: Slope = -1.64
Functional Effectiveness of Two-Party vs. Three-Party
When All SMs of Each Type Can Fail

Measures the proportion of time the Service User possesses the operational set of remote services needed to accomplish its task during $D$. 

- Two Party
- Three Party (3 SCMs)
- Three Party (2 SCMs)
- Three Party (1 SCM)

Notes:
1. Replacement case
2. 30 repetitions per data points
Presentation Outline

- One-Page Review of Project Objective and Plan
- One-Page Refresher on Service-Discovery Protocols
- Summary of Accomplishments on the Project (to date)
- An Autonomic Failure-Detection Algorithm for Distributed Systems
  - Applied to Service Registration in the Service Location Protocol (SLP)
  - Applied to Jini Leasing (SEE DEMO on WEDNESDAY EVENING)
- Performance of Service-Discovery Systems under Node Failure
- Plan for Final Six Months
- Conclusions
Plan for the Final Six Months

ONE IMPLEMENTATION, THREE CONFERENCE PAPERS, AND TWO JOURNAL PAPERS

- Implement our autonomic failure-detection algorithm for registration refresh and for directory polling in the meshSLP implementation available from Columbia University

- Write and submit a paper to WOSP 2004 on “An Autonomic Failure-Detection Algorithm for Distributed Systems”

- Write and submit two conference papers on failure-response in SLP (one covering message loss and communication failure and one covering node failure and power failure restart)

- Formalize a generic model of service-discovery architectures, including structure, behavior, and properties – write and submit a journal paper

- Write and submit a journal paper that characterizes the failure response of three service discovery protocols: Jini, UPnP, and SLP under hostile and volatile conditions
Conclusions

BY PROJECT’S END:

- We will have characterized performance and failure response for the three most widely accepted first-generation service discovery protocols (UPnP, SLP, and Jini), published our findings, and released the simulation models we used and data we collected.

- We will have devised and investigated three self-adaptive algorithms: autonomic failure detection (applied to various aspects of service-discovery protocols), inverted leasing, and adaptive jitter control for multicast search, published our findings, and released the simulation models we used and data we collected.

- We will have implemented, demonstrated, and validated our autonomic failure detection algorithm, as applied to Jini leasing, to SLP service registration and refresh, and to polling in Jini and SLP, published our findings, and released the implementations we used and data we collected.

- We will have constructed, implemented, tested, and verified a formal generic model of service discovery protocols that:
  -- encompasses all the functions and features of Jini, UPnP, and SLP
  -- defines consistency conditions that such protocols should satisfy
  -- identifies missing functions and other weaknesses in existing service-discovery systems and proposes improvements
  -- incorporates the self-adaptive algorithms we developed.