An Autonomic Failure-Detection Algorithm*

K. Mills, S. Rose, S. Quirolgico
NIST
Mail Stop 892
Gaithersburg, Maryland 20899
1-301-975-3618
{kmills, scottr, steveq}@nist.gov

M. Britton
Southern Methodist University
1265 Columbine Lane
Salina, Kansas 67401
1-214-641-9914
mbritton@mail.smu.edu

C. Tan
Montgomery Blair High School
403 Branch Drive
Silver Spring, MD 20901
1-301-593-8132
cetan@mbhs.edu

ABSTRACT
Designs for distributed systems must consider the possibility that failures will arise and must adopt specific failure detection strategies. We describe and analyze a self-regulating failure-detection algorithm that bounds resource usage and failure-detection latency, while automatically reassigning resources to improve failure-detection latency as system size decreases. We apply the algorithm to (1) Jini leasing, (2) service registration in the Service Location Protocol (SLP), and (3) SLP service polling.

1. INTRODUCTION
Recent research on failure detection and recovery in distributed systems reports non-functional periods comprising two distinct phases: periods when a system is unaware of a failure (failure-detection latency) and periods when a system attempts to recover from a failure (failure-recovery latency)[1]. Depending on system architecture and assumptions about failure characteristics of components, the study found failure-detection latencies covered from 55% to 80% of non-functional periods. The study also revealed failure detection can consume substantial overhead. These findings suggest distributed systems could benefit from failure detection algorithms that exhibit definite bounds on latency and overhead. We define and analyze such an algorithm, and then apply it to Jini leasing and to service registration and polling in SLP.

2. AUTONOMIC FAILURE-DETECTION
Figure 1 illustrates a two-way heartbeat failure-detection technique, where N *monitorables* each issue a rising heartbeat (every H_P seconds) to one *monitor*, which replies with a falling heartbeat. Assuming rising and falling heartbeat messages of known size (S_R and S_F, respectively), the system consumes network bandwidth $B = N\,(S_R+S_F)/H_P$. The monitor must process N/H_P heartbeat messages per second. The monitorable must process $1/H_P$ heartbeat messages per second.

Figure 2 defines the period of inconsistency when a monitorable fails between heartbeats. Should a monitorable fail immediately after receiving a falling heartbeat from a monitor, then the maximum failure-detection latency (L_{MAX}) is defined by the heartbeat period, i.e., $L_{MAX} = H_P$. Assuming a monitorable is equally likely to fail at any time, the average failure-detection latency (L_{AVG}) is half the heartbeat period, $L_{AVG} = H_P/2$.

*This work is a contribution of the U.S. Government, not subject to copyright. In addition, the work identifies certain commercial products and standards to describe our study adequately. The National Institute of Standards and Technology neither recommends nor endorses these products or standards as best available for the purpose.

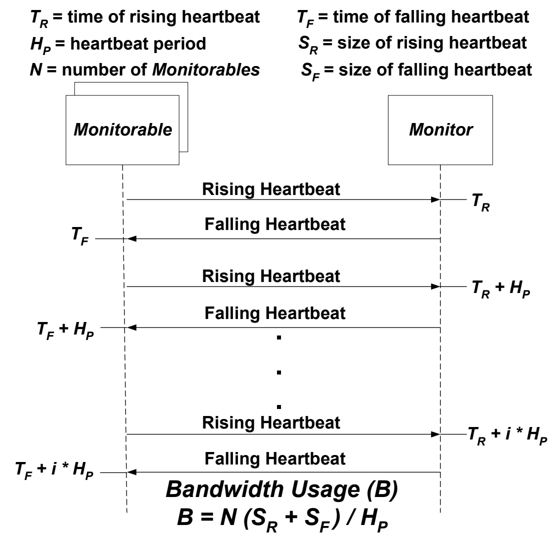
T_R = time of rising heartbeat
T_F = time of falling heartbeat
H_P = heartbeat period
S_R = size of rising heartbeat
N = number of *Monitorables*
S_F = size of falling heartbeat
Monitorable
Monitor
Rising Heartbeat
Falling Heartbeat
T_R
T_F
$T_R + H_P$
$T_F + H_P$
$T_R + i * H_P$
$T_F + i * H_P$
Bandwidth Usage (B)
$B = N\,(S_R + S_F) / H_P$

Figure. 1. Fundamental outlines of a two-way, heartbeat-based, failure-detection technique.

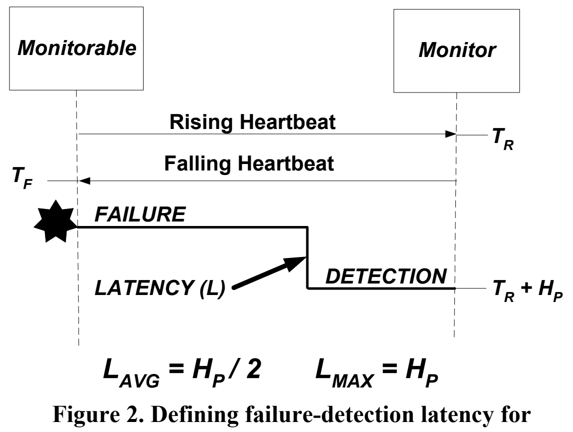

Figure 2. Defining failure-detection latency for heartbeat-based failure-detection techniques.

For monitor failure, where detection occurs when the monitor does not respond with a falling heartbeat, the situation differs slightly. A monitorable may wait for some timeout period (T_O) before concluding the monitor has failed; thus, the maximum detection latency for monitor failure is $H_P + T_O$, and the average detection latency is $H_P/2 + T_O$.

We define an autonomic algorithm to limit bandwidth usage to an allocated capacity (B_A) and to limit average failure-detection latency (to L_{WORST}), while reducing average failure-detection latency ($L_{AVG} < L_{WORST}$) to some lower bound (L_{BEST}) when the number of monitorables (N) falls below system capacity (N_{MAX}). We modify the two-way heartbeat technique so that the monitor includes a heartbeat period (H_P) in each falling heartbeat. The monitorable uses H_P to determine when to issue the next rising heartbeat. The monitor may vary H_P with each falling heartbeat to maintain an operating range defined by three policy goals: the average failure-detection latency in the worst (L_{WORST}) and best (L_{BEST}) cases and the allocated bandwidth (B_A).

```
if new monitorable then $N$++;
$H_P$ = N / C;
if $H_P > H_{MAX}$
        then $N$--;
                raise capacity exception;
        elseif $H_P < H_{MIN}$
                then $H_P = H_{MIN}$;
        endif
endif
```

Figure 3. Autonomic algorithm to vary heartbeat period.

Assuming N monitorables, the monitor varies the heartbeat period ($H_{MIN} \leq H_P \leq H_{MAX}$) using the algorithm in Figure 3.The maximum heartbeat period (H_{MAX}) is set to twice the worst-case average failure-detection latency ($H_{MAX} = 2\ L_{WORST}$). Given H_{MAX} and the monitor's capacity [$C = B_A/(S_R + S_F)$], a monitor can watch at most $N_{MAX} = H_{MAX}\, C$ monitorables. Assuming a monitor watches at least one monitorable, a natural choice for H_{MIN} would be $1/C$; however, this heart rate might place too great a load on individual monitorables. Instead, we choose a best-case goal (L_{BEST}) for average failure-detection latency and set $H_{MIN} = 2\ L_{BEST}$. H_{MIN} establishes a lower bound on failure-detection latency.

Below, we report analytical results as time-series plots (time increases from 0 to 400) with N first increasing to 200, and then decreasing back to 0. In all plots (Figure 4) we assume the same heartbeat sizes (S_R = 128 bytes and S_F = 64 bytes) and policy goals (L_{WORST} = 30 s, L_{BEST} = 7.5 s, and B_A = 576 bytes/s). The top plot shows our algorithm limits monitor workload to N_{MAX}, while the second plot illustrates our algorithm adjusting H_P between H_{MIN} and H_{MAX} as N varies. The third plot shows how average bandwidth increases and decreases but never exceeds allocated bandwidth (B_A). The bottom plot illustrates how our algorithm improves average failure-detection latency (L) as N decreases (ticks 200 to 400), while average failure-detection latency never exceeds the worst (L_{WORST}) and best (L_{BEST}) cases.

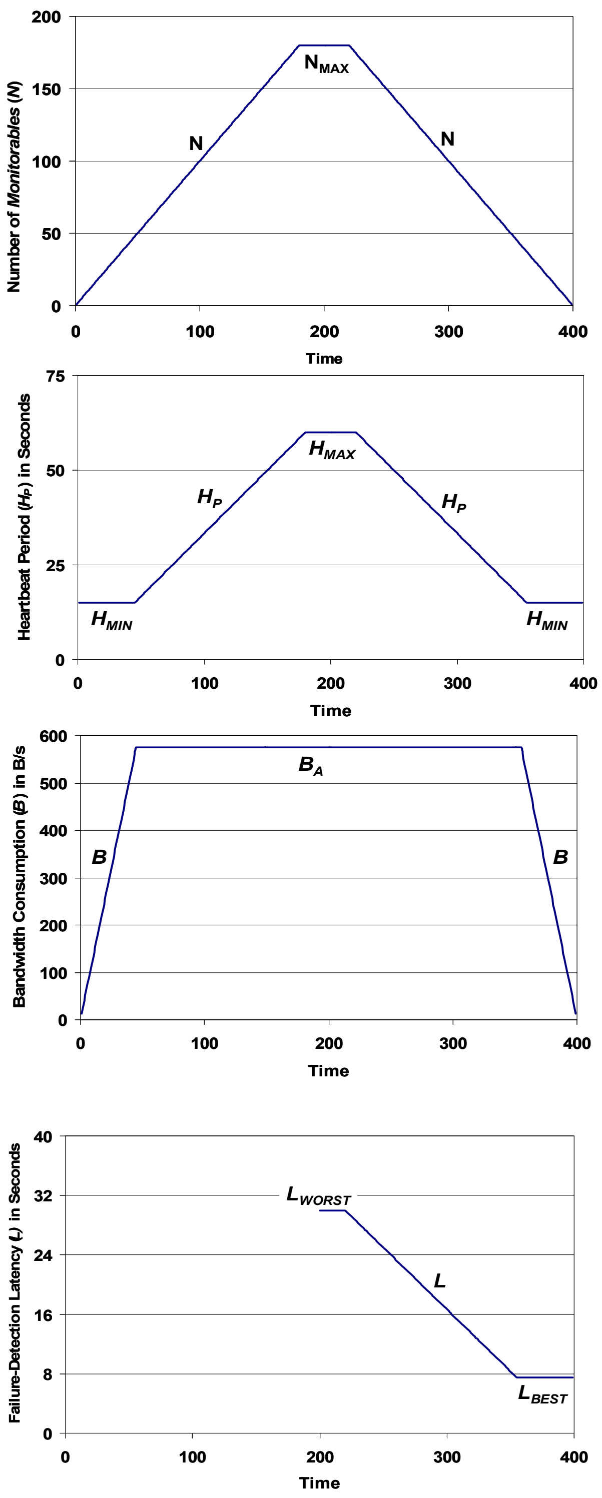

Figure 4. Time-series plots from an analytical model of the proposed autonomic failure-detection algorithm.

3. SAMPLE APPLICATIONS

We apply our algorithm to selected functions in two service-discovery protocols: Jini [2] and the Service Location Protocol (SLP) [3]. ***Jini*** enables clients and services to rendezvous through a third party, known as a *lookup service*. Each Jini service registers a description of itself with each discovered lookup service, and requests a lease duration (L_R), which may be accepted at time T_G for a granted lease period $L_G \leq L_R$. L_R may be "any", which allows a lookup service to assign any value for L_G. To extend registration beyond L_G, services must renew the lease prior to an expiration time $T_E = T_G + L_G$; otherwise, registration is revoked. This cycle continues until a service cancels or fails to renew a lease. Lookup services assign L_G within a configured range, $L_{MIN} \leq L_G \leq L_{MAX}$. While a granted lease may not be revoked prior to T_E, lookup services may deny any lease request.

We apply our algorithm to enable Jini lookup services to vary L_G within a bounded range ($L_{MIN} \leq L_G \leq L_{MAX}$) while limiting resource consumption associated with lease renewal. The mapping is straightforward. Assuming three policy goals, bandwidth allocated (B_A) and worst (L_{WORST}) and best (L_{BEST}) failure-detection latencies, we compute $L_{MIN} = H_{MIN} = 2\ L_{BEST}$ and $L_{MAX} = H_{MAX} = 2\ L_{WORST}$. Knowing the size of the rising (lease request) and falling (lease grant) heartbeats (S_R and S_F, respectively), leasing capacity (C) is computed as before. Knowing the number of registered services (N), a Jini lookup service uses the algorithm in Figure 3 to compute H_P and then uses that value as the granted lease period ($L_G = H_P$). If the new lease would exceed system capacity, then the lookup service issues a *LEASE_DENIED* exception.

To verify our analysis, we implemented our algorithm in a Jini simulation and compared simulation results against analytical predictions, given a selected set of policy goals and known sizes for Jini messages. We subsequently implemented our algorithm in a publicly available implementation of the Jini lookup service. We modified the lookup service code to accept our policy goals and to measure and report average bandwidth usage (B), the number of registered services (N), and the value for L_G. We deployed our modified Jini lookup service in a test bed built to control and monitor thousands of Jini services. We coded a measurement client to detect service arrivals and departures, computing average failure-detection latency (L). We measured behavior of a live Jini system using the same policy goals selected for analysis and simulation. We report our results in Figure 5 as four time-series plots, where we used the same protocol parameters (S_R = 350 bytes and S_F = 350 bytes) and policy goals (B_A = 2100 bytes/second, L_{BEST} = 7.5 seconds and L_{WORST} = 1200 seconds, and so N_{MAX} = 7200) for the analysis, the simulation (1000 repetitions per data point), and the live system (20-30 repetitions per data point).

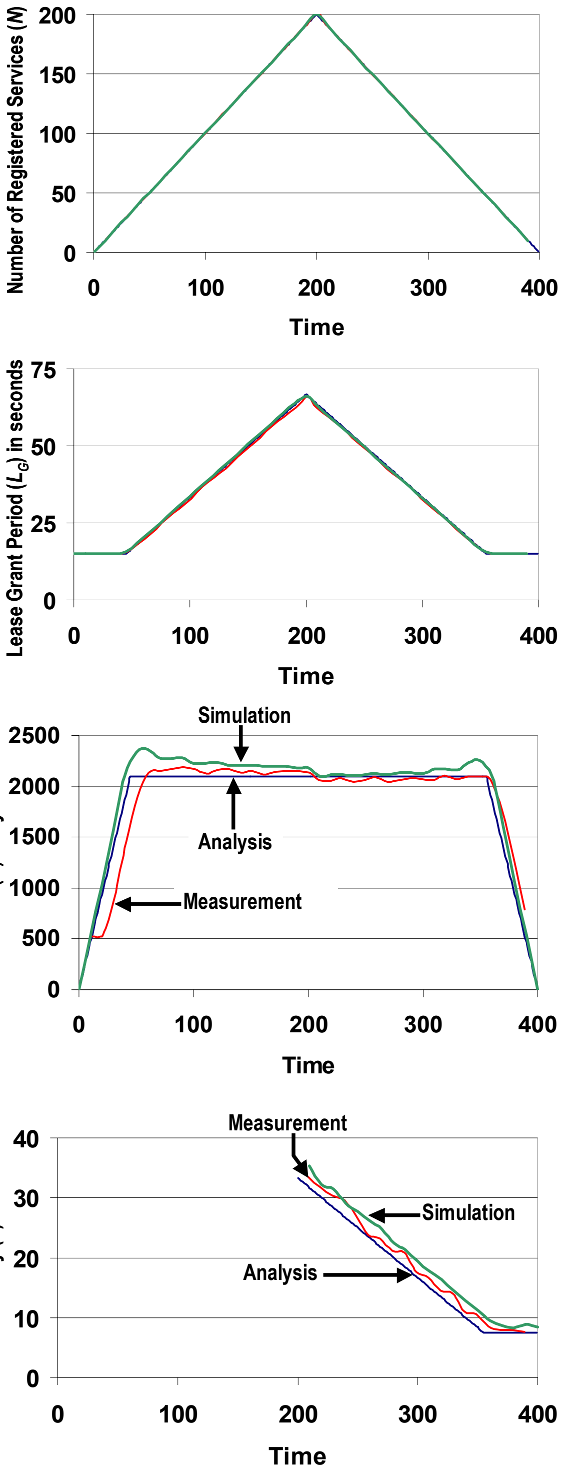

Figure 5. Time-series plots showing application of autonomic failure-detection to Jini leasing procedures

SLP Service Registration. SLP enables clients, called user agents (UAs), and services, called service agents (SAs), to rendezvous through a third party, known as a directory agent (DA). A SLP SA registers a description of itself with each discovered DA. A UA may query any discovered DAs to find services of interest and to obtain attributes that describe services.

A SA requests registration for a time-to-live (TTL_R), which may be accepted by a DA at time T_G. To extend registration beyond TTL_R, the registering SA must renew the registration prior to an expiration time $T_E = T_G + TTL_R$; otherwise, the DA revokes the registration. This cycle continues until the SA cancels or fails to refresh a registration. While an accepted registration may not be revoked prior to T_E, a DA may deny any registration request. A DA will always deny a registration request when TTL_R is too small, as determined by comparing the TTL_R against a minimum-refresh interval (RF_{MIN}) included within advertisements multicast by the DA at a periodic rate (DA_{BEAT}). We apply our algorithm to provide SLP service registration with rapid feedback, eliminating the need for RF_{MIN}.

We add a field (TTL_G) to the *SrvAck* message, used by DAs to acknowledge service-registration (*SrvReg*) messages from SAs. Then, a DA can ignore TTL_R and instead compute a granted time-to-live (TTL_G), which can vary dynamically within a bounded range ($TTL_{MIN} \leq TTL_G \leq TTL_{MAX}$) while limiting resource consumption associated with refreshing service registrations. The mapping is straightforward. Assuming our three policy goals, bandwidth allocated (B_A) and worst (L_{WORST}) and best (L_{BEST}) failure-detection latencies, we compute $TTL_{MIN} = H_{MIN} = 2\ L_{BEST}$ and $TTL_{MAX} = H_{MAX} = 2\ L_{WORST}$. Knowing the size of the rising (*SrvReg*) and falling (*SrvAck*) heartbeat messages (S_R and S_F, respectively), registration capacity (C) is computed as before.

Knowing the number of registered services (N), a DA can use the algorithm in Figure 3 to compute H_P and then use that value as the granted time-to-live ($TTL_G = H_P$). If a new registration would exceed system capacity, then the DA issues the *SrvAck* with a status code of *DA_BUSY_NOW*.

To verify our analysis, we implemented our algorithm in a SLP simulation and compared simulation results against analytical predictions. We report our results in Figure 6 as four time-series plots, where we used the same protocol parameters (S_R = 76 bytes and S_F = 56 bytes) and policy goals (B_A = 396 bytes/s, L_{BEST} = 7.5 seconds and L_{WORST} = 500 seconds) for the analysis and the simulation. Setting L_{WORST} = 500 seconds and B_A = 396 bytes/s provides a maximum system capacity of N_{MAX} = 3000 registered services. In the main, Figure 6 shows a close correspondence between analytical predictions and simulation results; however, the bandwidth-usage simulation plot (as well as the bandwidth-usage simulation plot for Jini leasing procedures - recall Figure 5) illustrates a hysteresis within the control loop of our proposed algorithm. During periods of increasing system size, the algorithm typically assigns a heartbeat period that immediately becomes too small for the now increased system size, and will only be able to reduce the heartbeat period one monitorable at a time, as each previously assigned heartbeat expires. This lag causes the algorithm to slightly overshoot the allocated bandwidth. The larger the heartbeat message size the greater the overshoot. For example, the Jini plot (700 bytes per heartbeat) overshoots allocated bandwidth more than the SLP plot (132 bytes per heartbeat). However, the downward slope in the bandwidth-usage simulation plots (as system size increases from 50 to 200) suggests that the algorithm will stabilize bandwidth usage at the allocated bandwidth once the system size stabilizes.

SLP UA Polling. SLP UAs must poll DAs periodically to learn about service arrivals and departures or about changes in attribute values of service descriptions. SLP includes no mechanisms through which DAs can control the polling rate of UAs. We can modify DA procedures to determine which UAs are polling a DA, and then we can apply our algorithm to assign polling intervals to those UAs. First, we explain the modified DA procedures.

When a UA queries a DA, either using a service-request (*SrvRqst*) or attribute-request (*AttrRqst*) message, we modify the DA procedures to lookup the UA in a local DA cache. If the UA is not found, then the DA creates a new cache entry for the UA; otherwise, the DA uses the existing cache entry. The DA grants the cache entry a polling interval (P_G), which can vary dynamically within a bounded range ($P_{MIN} \leq P_G \leq P_{MAX}$) while limiting resource consumption associated with UA polling. We modify the format of the appropriate reply message, either the service reply (*SrvRply*) or the attribute reply (*AttrRply*), to include a field to hold P_G for return to the UA. If the UA fails to issue another query to the DA by the time P_G expires, then the DA purges the associated entry from the local cache of UAs. Upon receiving P_G in the reply message, the UA schedules its next poll (if any) of the DA to occur slightly before P_G expires.

The DA can use our algorithm to determine a suitable value for P_G. The mapping is straightforward. Assuming our three policy goals, bandwidth allocated (B_A) and worst (L_{WORST}) and best (L_{BEST}) failure-detection latencies, we compute $P_{MIN} = H_{MIN} = 2\ L_{BEST}$ and $P_{MAX} = H_{MAX} = 2\ L_{WORST}$. Estimating an average size for the rising (*SrvRqst* or *AttrRqst*) and falling (*SrvRply* or *AttrRply*) heartbeat messages (S_R and S_F, respectively), registration capacity (C) is computed as before. Knowing the number of polling clients

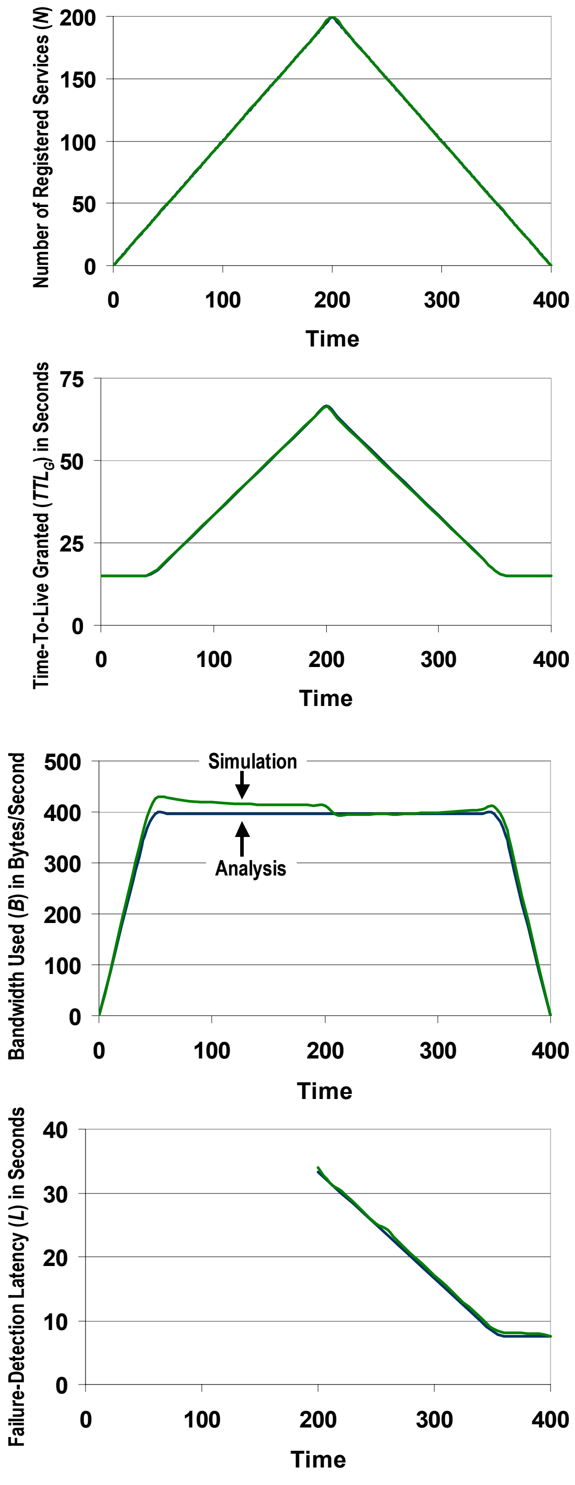

Figure 6. Time-series plots showing application of autonomic failure-detection to SLP service-registration refresh procedures

(N), a DA can use the algorithm in Figure 3 to compute H_P and then use that value as the assigned polling interval ($P_G = H_P$). If a new polling UA would exceed system capacity, then the DA issues the *SrvRply* or Attr*Rply* with a status code of *DA_BUSY_NOW*.

To verify our analysis, we implemented our algorithm in a SLP simulation and compared simulation results against analytical predictions, given a selected set of policy goals and estimated sizes for SLP messages. We devised a specific polling algorithm. Upon discovering a DA, a UA first issues one *SrvRqst* message (receiving a *SrvRply* from the DA) and then an *AttrRqst* message (receiving a *AttrRply* from the DA). The DA grants a P_G only for each *AttrRply* message, and the UA polls only with *AttrRqst* messages. In other words, on initial discovery a UA and DA exchange four messages (*SrvRqst-SrvRply-AttrRqst-AttrRply*), and then the UA and DA periodically exchange two messages (*AttrRqst-AttrRply*). We modified our analytical model to account for these polling procedures.

We report our results in Figure 7 as four time-series plots, where we used the same protocol parameters (average S_R = 77 bytes and average S_F = 128 bytes) and policy goals (B_A = 615 bytes/s, L_{BEST} = 7.5 seconds and L_{WORST} = 500 seconds) for the analysis and the simulation (1000 repetitions per data point). Setting L_{WORST} = 500 seconds and B_A = 615 bytes/s provided a maximum system capacity of N_{MAX} = 3000 registered services. For the simulation, we sampled individual message sizes from a distribution for each *AttrRqst* and *AttrRply*. The distribution parameters for an *AttrRqst* were: 4 bytes minimum, 256 bytes maximum, 77 bytes average, and 138 bytes variance. The distribution parameters for an *AttrRqst* were: 64 bytes minimum, 224 bytes maximum, 128 bytes average, and 25 bytes variance.

Figure 7 shows a close correspondence between analytical predictions and simulation results, and also again illustrates the hysteresis associated with the bandwidth-allocation control loop. Here, the overshoot is worse than for SLP service registration because the SLP polling heartbeat message sizes are greater (205 bytes on average compared with 132 bytes). The bandwidth-usage overshoot is lower than for Jini leasing, however, because the SLP polling heartbeat messages are smaller (205 bytes on average compared with 350 bytes). Further, the overshoot for SLP polling is somewhat exaggerated because the initial four-message exchange prior to the polling heartbeats is included in the bandwidth-usage during periods of increasing system size. This can also be seen in the results from our modified analytical model, which overshoots the allocated bandwidth (615 bytes/s). This effect diminishes as the granted polling interval (P_G) increases, as can be seen in the downward slope in both the analytical predictions and simulation results.

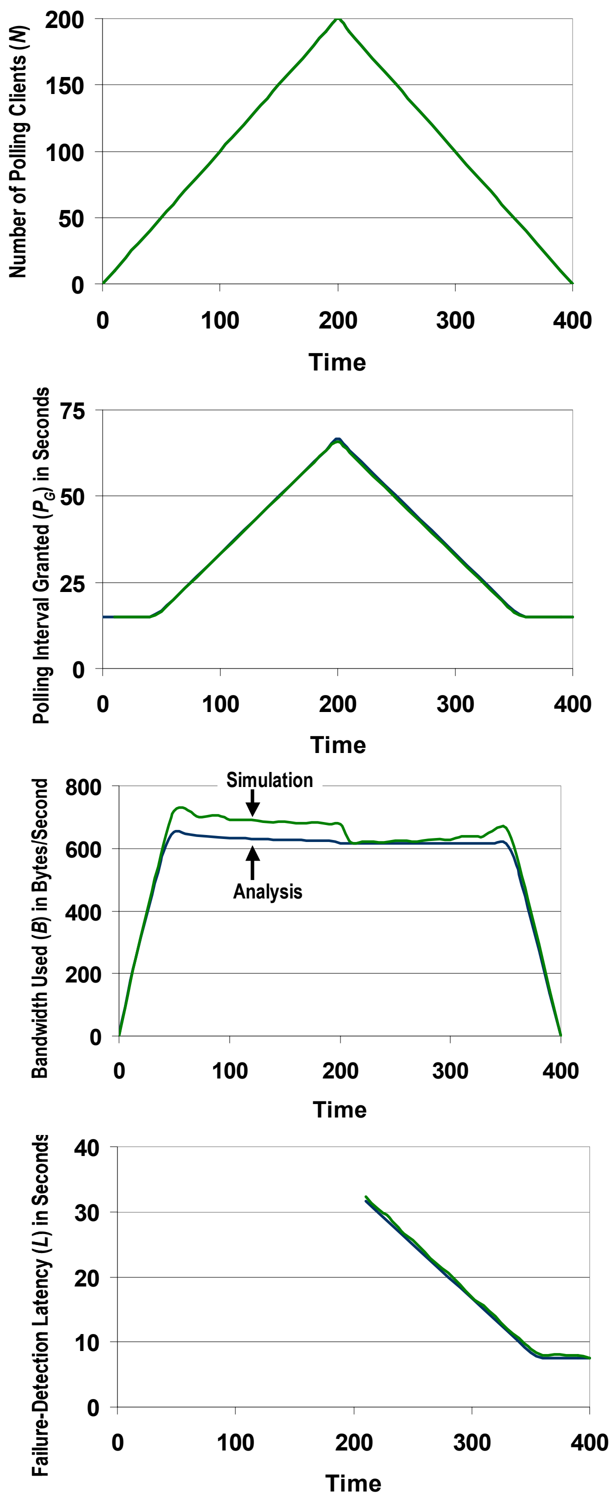

Figure 7. Time-series plots showing application of autonomic failure-detection to SLP UA polling

4. REFERENCES

[1] C. Dabrowski, K. Mills, and A. Rukhin A. Performance of Service-Discovery Architectures in Response to Node Failures, *Proceedings of the 2003 International Conference on Software Engineering Research and Practice* (SERP'03), CSREA Press, June 2003, pp. 95-101.

[2] K. Arnold et al, The Jini Specification, V1.0 Addison-Wesley 1999. Latest version is 1.1 available from Sun.

[3] Service Location Protocol Version 2, Internet Engineering Task Force (IETF), RFC 2608, June 1999.