# Ratatoskr: Wide-Area Actuator RPC over GridStat with Timeliness, Redundancy, and Safety

Erlend S. Viddal, Stian Abelsen, David E. Bakken and Carl H. Hauser

Technical Report EECS-GS-011
Washington State University
School of EECS
Pullman, WA 99163

December 2007

# Ratatoskr: Wide-Area Actuator RPC over GridStat with Timeliness, Redundancy, and Safety

Erlend S. Viddal*, Stian Abelsen†, David E. Bakken‡ and Carl H. Hauser
Washington State University
School of Electrical Engineering and Computer Science
Pullman, Washington, USA
{eviddal, stian.abelsen}@gmail.com, {bakken, chauser}@wsu.edu

## Abstract

*GridStat is a QoS-managed middleware framework designed to replace the power grid's aging, inflexible, and slow data communications system. GridStat is a specialization of the publish-subscribe paradigm that takes advantage of the semantics of periodic status data updates to provide data delivery with a per-subscriber rate, latency, and redundant paths. While GridStat is well-suited for delivery of sensor data over a wide-area network, its baseline one-way mechanisms are not suitable for round-trip invocations such as setting an acutator or calling between control centers. In this paper we present the design, implementation and experimental evaluation of Ratatoskr, a tunable RPC mechanism that builds on the QoS semantics of GridStat and supports three kinds of redundancy. Additionally, user-defined pre- and post-condition predicates over GridStat status variables are built into the call semantics. Preconditions can abort an RPC call if the measured conditions dictate, while post-conditions provide additional physical assurance the call succeeded.*

**Keywords**: Remote procedure call, redundancy, quality of service, safety

**Submission Category**: Regular Paper

## 1. Introduction

The communication infrastructure of the electric power grids in the USA were developed largely as a result of the 1965 blackout. Due to low investment by utilities, this infrastructure has not been meaningfully updated since then; it is still slow, inflexible, and does not even meet today's data delivery requirements let alone tomorrow's [9, 5, 3].

GridStat is a QoS-managed publish-subscribe middleware framework designed to augment and eventually replace the grid's aging communication system [6, 3, 7]. It provides one-way delivery with per-subscriber rate, latency, and redundant disjoint paths over a critical infrastructure's wide-area network. GridStat has been involved with a number of research collaboraitons with electric utilities since 2004 and has been under development since 2000.

GridStat's one-way delivery mechanisms are sufficient for delivering updated values from remote sensors, However, they are inadequte for a round-trip remote procedure call (RPC), for example to an actuator in the power grid or between control centers. There has been some work in CORBA on real-time and fault-tolerance properties [18, 13, 17, 16]. However, none of these was designed specifically for the kinds of conditions faced in a wide-area network [22], and we believe the CORBA object model is far to complex for the more modest needs of a simple request-reply call to an actuator [20].

In this paper we present Ratatoskr[1], a highly tunable RPC mechanism tailored to the needs of the power grid and built over GridStat. The research contributions of this paper are:

- Design and implementation of an RPC mechanism over QoS-managed, one-way publish subscribe mechanisms. The RPC mechanism supports three distinctive techniques for redundancy, offering tradeoffs between worst-case deadlines, use of network resources and resiliency towards a variety of network failures. Applications are allowed fine control of redundancy semantics.

*Current affiliation: Simula Innovation, Oslo Norway
†Current affiliation: Eltek Valere, Drammen Norway
‡Contact author

[1] In Norse mythology, Ratatoskr is a squirrel climbing around the great world tree Yggdrasill, ferrying messages and gossip between the mythological creatures living in its branches, and in particular delivering insults between an eagle residing in the treecrown and a dragon gnawing on the roots.

- Design and implementation of pre- and post-condition mechanisms integrated with RPC semantics provides additional functionality when compared to application-level implementations.
- Experimental evaluation quantifying the redundancy techniques.

The remainder of this paper is organized as follows. Section 2 gives an overview of GridStat. Section 3 describes the Ratatoskr architecture, while Section 4 details its RPC mechanism. Section 5 presents an experimental evaluation of Ratatoskr. Section 6 overviews related work, while Section 7 concludes and discusses future research.

## 2 GridStat

GridStat is a QoS-managed, publish-subscribe middleware framework designed for delivering periodic updates of sensor data for the electric power grid and other wide-area critical infrastructures. Its architecture is given in Figure 1. GridStat's *data plane* is responsible for delivering the sensor data. A single sensor datum is called a *status variable*, and can be subscribed to by multiple subscribers. The data plane consists of *pubishers*, which produce the sensor data; *subscribers*, which consume the data; and a network of *status routers*, specialized middleware-layer forwarding engines that deliver the status variables.

GridStat's *management plane* is responsible for managing the data plane, including making admission control decisions. It is organized into a hierarchy of *QoS Brokers* that is designed to map directly onto the geographically-based hierarchies that the power grid and other critical infrastructures are normally organized into. A *leaf QoS broker* is the lowest level in this hiearchy and is responsible for, and directly connected to, a collection of status routers that are organized into a single administrative domain called a *cloud*.

GridStat's data delivery semantics are novel in two broad ways. First, they are a specialization of the publish-subscribe paradigm tailored to leverage the semantics of periodic streams of data updates. For example, in a normal publish-subscribe system, an event that is being forwarded from a publisher to its subscribers cannot be arbitrarily dropped, because it is in general impossible to know how it will affect the application programs that subscribe to it. However, in GridStat the subscribers' required rate, latency, and number of redundant (disjoint) paths are part of the API to subscribe to a given status variable[6, 10]. With this knowledge, a status router can discard updates, an operation we call *rate filtering*, when downstream subscribers have gotten an update recently enough to satisfy their latency and rate requirements. This can potentially save a large amount of bandwidth, because many sensors in the power grid produce updates with a hard-coded rate that is

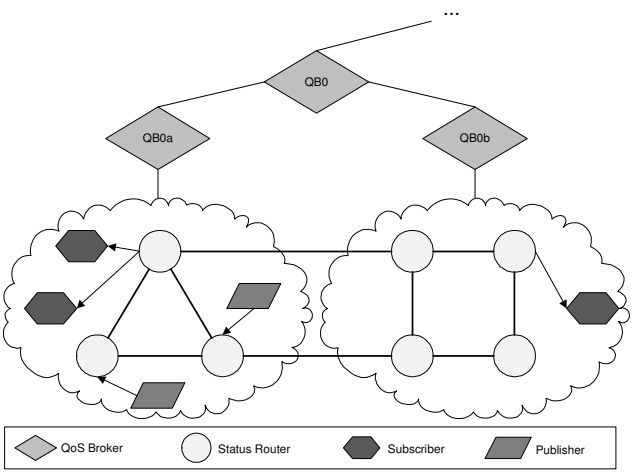


**Figure 1. GridStat architecture**

deliberately conservative, *i.e.* at a rate above what its engineers believed any application would require. Rate filtering in GridStat preserves temporal relationships between different GPS-timestamped updates; for more details, see [11, 6].

The second way in which GridStat's data delivery is novel is that different subscribers to a given status variable can be provided with different rate, latency, and number of redundant paths. This allows a power device such as a relay that is physically very close to the publisher to be given a high rate and low latency, while remote subscribers that only need to loosely track the variable can be given a much lower rate and higher latency. If this flexibility was not supported, then the remote subscribers would have to receive the high rate of delivery that the device in the same substation is given. This can be a large amount of bandwidth over a large geographical area: for example, many hundreds of miles in a large power grid such as that in Western Europe or the Western USA.

GridStat has been under development since 2000 [2], and has been receiving live data feeds from a regional electric utility since 2004 [21]. It has been involved in a regional pilot project involving two US national energy labs since mid-2007, and a pilot project involving utilities in the Southeastern USA is planned for early 2008. More details on the architecture and design of GridStat can be found in [6, 3, 7].

## 3 Ratatoskr

Ratatoskr uses a two-tiered architecture. A communication module implements the 2WoPS protocol, providing timely, fault-tolerant two way communication over one-way GridStat paths; and an RPC module provides RPC server and client functionality. Separating the communication protocol into a separate module allows the communication

module to be reused by other applications. For example, the 2WoPS protocol has been utilized for control communication between GridStat QoS brokers [1].

Figure 2 shows an overview of the modules of Ratatoskr (light shade), the GridStat modules used by Ratatoskr (dark shade), and other applications using the modules (white). The example shows the architecture stack for a control center and a substation. The main intended use of Ratatoskr is illustrated by the control center's energy management system (EMS) using Ratatoskr RPC to execute control operations on an actuator in the substation. We believe that it also can be useful for control messages going to other targets than an actuator, for example a peer control center or from a higher-level balancing authority to a control center, but a detailed investigation of its applicability to these contexts is future work.

### 3.1. The 2WoPS Protocol

To achieve a two-way communication style needed for RPC signaling, Ratatoskr uses two GridStat pub-sub connections. Each RPC host utilizes both a publisher and a subscriber - the publisher for sending data as published status variables and the subscriber for receiving data. Two hosts wishing to communicate each publish a new status variable and subscribe to the other hosts published variable. Data is sent over the connection by publishing status events. It should be noted that as GridStat uses the status dissemination pub-sub paradigm for message forwarding, sending data in this manner has a very low routing overhead, as paths are established at connection setup time and forwarding ports are mapped to a simple connection identifier. Currently the 2WoPS protocol supports only communication between two hosts, but the GridStat network has excellent support for multicast of status events so future versions could add support for one-to-many and group communication styles. For the remainder of the paper, the initiator of a connection will be referred to as a 2WoPS client and a receiving host the 2WoPS server, although in practice no distinction is made after connection setup.

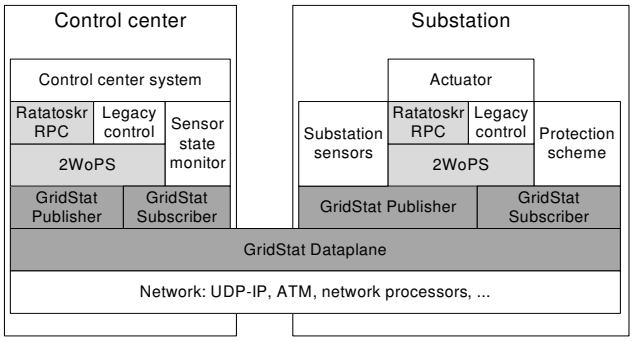


**Figure 2. Ratatoskr module stack**

The 2WoPS protocol leverages the QoS guarantees provided by the GridStat publish subscribe paths to supply timely communication. Further, the protocol aims to provide a high degree of fault tolerance while retaining predictable delivery times. To this end, three redundancy techniques are employed to strengthen delivery guarantees:

- *Spatial redundancy:* One major feature of GridStat is that subscriptions may be routed through multiple disjoint network paths. If a status event sent over two redundant paths is dropped or corrupted along one path, successful delivery is still possible along the secondary path. The 2WoPS protocol exposes the number of disjoint paths used for the paths between the 2WoPS client and server and for the return path during setup connection time, and filters any excess status events upon receipt. For a connection in which both directions employ $n$ disjoint paths, we say that the connection has a *spatial redundancy level* of $n$. Status events sent over such a connection will tolerate $n-1$ losses, and are not affected by temporal concentration of errors. It should be noted that common mode failures thoughout the network, such as very high traffic levels during attacks or crisis situations may affect components involved in all redundant paths. The degree of spatial redundancy possible for a connection depends on the network topology as the network must support disjoint paths from the 2WoPS client to the 2WoPS server, but it is expected that the high reliability requirements of a critical infrastructure as the power-grid will justify the expense of building a network with a high degree of redundancy. Spatial redundancy incurs some delay overhead because of routing complexity and, also depending on the network topology, as the delay-optimal route might be incompatible with provoding the required number of disjoint paths. It should be noted that the process of allocating redundant paths through the network is more complex than allocating a single path, and so spatial redundancy induces overhead to the network management, but only at subscription setup time. GridStat supports delay guarantees also for paths redundant to the best-delay route (the *primary path*), and so a connection employing $n$ spatial redundancy will provide delay guarantees even in the face of up to $n-1$ losses. Redundant disjoint paths also lower the expected delivery latency in the face of occasional dropped or delayed packets.

- *Temporal redundancy:* While spatial redundancy provides protection from a range of network failures, the network costs and delay properties of a large number of paths could prove prohibitive in cases where a very high degree of fault tolerance is required or where

the network topology is unsuited for spatial redundancy. Ratatoskr offers temporal redundancy by allowing each data unit sent by the 2WoPS protocol to be repeatedly published without waiting for a round-trip. A data unit published $n$ times has a *temporal redundancy level* of $n$ and tolerates $n - 1$ losses. The technique consumes $n$ times more bandwidth than regular sends, but has no additional setup costs. Additionally, in comparison to spatial, which has significant worst-case delay overhead because of longer paths, and ACK/retry, which may be unbounded, temporal redundancy adds minimal overhead to the worst case delay (the send time of the copy of the packet). Temporal redundancy does not provide the same level of protection as spatial redundancy, as network losses may be temporally concentrated. For example, network congestion will often lead to periods of high loss rates when a router's buffer for an outgoing link is full or maintenance on a router might disable all connecting links for several minutes. To add to this, by sending several copies of the same 2WoPS packet in a short span, the extra bandwidth used might add to existing congestion. The 2WoPS protocol allows applications to specify a delay between sending each temporally redundant copy of a 2WoPS packet, and the degree of temporal redundancy may be specified for each send.

- *ACK/resend*: While spatial and temporal redundancy provides a high degree of fault tolerance, they do not provide any feedback to the application about successful delivery, and do not provide protection from failures that affect the whole network, such as periods of heavy congestion. Ratatoskr provides a third technique for redundancy by having message receivers ACK successfully received 2WoPS messages, and allowing the user to specify that messages that do not produce an ACK should be resent up to $n$ times. A message that is set to be resent up to $n$ times when ACKs are missing has an *ACK/retry level* of $n$. Successfully received ACKs are reported back to the server, while messages that do not receive an ACK even after $n$ resends are reported with *unknown delivery* status.

#### 3.1.1 Combining Redundancy Techniques

Redundancy measures can be combined as seen fit by the user. Temporal and spatial redundancy measures are cumulative and affect all 2WoPS packets, including ACKs. For example, if a sending a 2WoPS packet with 3 temporally redundant sends and 4 ACK/resends over a connection with 2 spatially redundant paths in each direction, three status events containing copies of the message packet will be sent. At the entry-point router, each of the status events will be forwarded to the first routers of the redundant paths, and, assuming no network failures, six status events containing the same 2WoPS packet will arrive at the receiver. The receiver will similarly send a single ACK 2WoPS packet, which will be sent in three temporally redundant status events, which again will be copied onto the redundant paths. If all ACKs are lost in the network, the sender will resend three new status events containing the message, and so on. This gives application designers the ability to tailor a connection to the needs of the application, allowing use of high spatial and temporal redundancy where a low delay is required, or relying on ACK/resend for redundancy for less delay-sensitive applications or where bandwidth is scarce. Combining redundancy techniques in this manner gives a delivery guarantee based on the cumulative worst cases of the redundancy techniques. A 2WoPS packet sent with spatial and temporal redundancy will have an end-to-end delivery guarantee equal to the last temporally redundant packet sent along the longest spatially redundant path. The timeout for retrying a packet sent with ACK/retry, spatial and temporal redundancy is calculated as two times the end-to-end delivery guarantee, as in the worst case the only status event containing the message that reaches the server is the last temporally redundant along the longest spatial path, and the same for the ACK.

## 4 The Ratatoskr RPC Mechanism

Ratatoskr RPC (RRPC) is a remote procedure call protocol for power-grid control communication built on top of the 2WoPS protocol. The primary application for Ratatoskr is remote operation of power-grid actuators, either to a gateway interfacing multiple devices or by directly controlling intelligent electronical devices (IEDs) with remote interfaces embedded in the actuator itself. The current grid communication system is unable to support trends in the power-grid stress levels and the security threat picture, and proposed solutions require real-time, reliable control [5].

RRPC draws extensively on the features provided by the 2WoPS protocol. Delivery guarantees for calls is achieved using GridStat's QoS enabled network communications. Because the use of redundancy trades off network resources, or worst case delay in the case of ACK/retry, against safety, applications designers are allowed detailed control over the techniques used for redundancy.

In addition to increased safety through fault-tolerance, pre- and post-conditions on calls are built into the RPC semantics. Pre-conditions are conditional expressions over GridStat status variables that are evaluated at the server (a power grid actuator) before execution of an RRPC call, and if the expression is false, the call to the physical actuator is not executed. Post-conditions are similar expressions evaluated after the call has executed, and negative results are reported back to the application via exception. This can

provide a physical confirmation that the actuator succeeded even if all of the redundant reply messages are lost.

### 4.1 Pre-and Post Conditions

For power grid control operations, placing pre-conditions on the execution may help in preserving safety in face of unwarranted situations such as power-grid anomalies or unexpected mechanical operation. Because of the distances involved, communication must be expected to suffer from bandwidth and delay limitations. Thus one may assume that the client-side information about server state is limited and not fully updated, and following this, pre-conditions should be placed on the server-side of the call.

Examples of pre-conditions in power-grid operation are:

- Isolators are actuators that connect and disconnect de-energized power circuits. A precondition could be to verify that a line is de-energized before attempting isolation. (A number of utility technicians are electrocuted every year even though both the EMS software and the control center operators were certain the line was de-energized.)
- High voltage equipment carries with it electrocution hazard, and another pre-condition could be to verify that no manned maintenance is scheduled at a field site when performing operations that might place maintenance personnel in danger.

Ratatoskr further allows application designers to use the same predicate modules used for pre-conditions to place post-conditions on calls. Power-grid operations are complex, and actuator operations may give unexpected results in face of situations such as mechanical malfunctions or operator overrides. Server-side post-conditions utilize the rich information environment local to the substation for analyzing the physical outcome of an execution and returning a brief report to the client. Thus client applications can review the results of calls without having to retrieve large amounts of data from the substation. By allowing a delay before the post-condition is evaluated, the effect of the operation is allowed to stabilize. By designing the post-conditions into the RPC semantics, the result of the post-condition is transferred to the client as a separate send, without affecting the delay of the RPC call itself.

Examples of post-conditions in power-grid operation are:

- Load tap changers are components of certain transformers that allow adjustmentment of output voltage. A post condition could be to verify the new voltage level after load tap changer operation, or even to generate a status report from all connected devices and send it back to the client.
- Transformer protection is a scheme to detect internal faults in a transformer and isolate it by breaking all connected lines if a fault is detected. A post-condition could be to trigger a transformer protection scheme after all transformer operations.

## 5 Evaluation

Ratatoskr was evaluated with respect to performance in face of network faults. The purpose is to understand the efficiency of the implemented fault tolerance techniques on RPCs over a faulty network, not to evaluate the performance of the prototype implementation, as a real-time implementation is outside the scope of the prototype design.

### 5.1 Evaluation Procedure

Evaluations were performed by connecting Ratatoskr client and server processes to a small GridStat network and commencing a number of RPC calls from the client to the server. To introduce network faults, event channels between status routers were routed through a network link emulator introducing delay and errors. Links going through the link emulator are referred to as *emulated links*. This setup tries to emulate control traffic over a wide area network of status routers, where both the client and server are either connected to their entry-point status router by a local area network, or running as processes on the same computer. No delay or loss is induced on the link between RRPC peers and their entry-point SRs. This is based on the expectation that a deployment of GridStat will include a wide deployment of status routers throughout the power grid to provide a high degree of network redundancy, which makes it likely that sites using RRPC also contain a status router.

#### 5.1.1 Evaluation Topology

The topology used for the evaluation is shown in figure 3. 13 status routers form two paths between the client and the server, one of 7 links and one of 6. This is to allow for the fact that when using spatial redundancy additional paths may often be longer than a single best path. The current implementation of GridStat does not allow more than two redundant paths. No additional links and status routers outside the two paths were employed as routing between the same two peers in a static network (such as the current version of GridStat) always uses the same path no matter the errors.

### 5.2 Evaluation Setup

#### 5.2.1 Processes

The processes used for testing were:

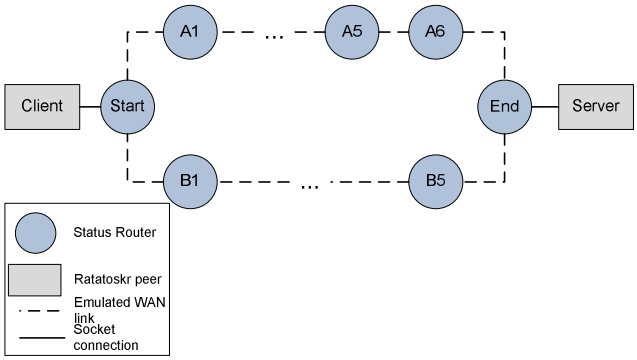


**Figure 3. Evaluation topology**

- 13 GridStat status routers
- 1 GridStat leaf QoS broker
- 1 evaluation program implementing a Ratatoskr client peer *(client)*
- 1 evaluation program implementing a Ratatoskr server peer *(server)*
- 1 Sun Java CORBA nameserver, not shown in Figure 3
- 1 Java program providing socket tunnels with loss and delay properties *(network link emulator)*, not shown in Figure 3

#### 5.2.2 Hardware

The evaluation was tested on a single computer, running a Core2 duo 2.13GHz dual-core processor and 4 gigabytes of RAM. The operating system was Ubuntu linux, kernel 2.6.20-16 compiled with 1 millisecond kernel tick intervals and full kernel preemption. All evaluated programs were implemented in Java, compiled and run with sun java2SE 6 (version 1.6.0.00). All inter-process communication was done over operating system UDP sockets for GridStat data traffic and Sun's Java 2 Platform CORBA Package for control. All tests were performed in user mode with regular process priority and with the graphical operating system interface turned off.

#### 5.2.3 Garbage Collection Handling

A mechanism was implemented to measure the impact of garbage collection and substract this from call results. The mechanism queries the Java `java.lang.management.GarbageCollectorMXBean` interface for recent garbage collection operations and their durations, and if gc operations have occured, substracts the gc operation duration from the end-to-end duration of the affected call. Early measurments revealed that the garbage collector had a significant impact on results. As the purpose of the evaluation was to evaluate the effectiveness of the redundancy measures, and not the execution-time predictability of implementation, we found that removing garbage collection would improve measurment data. Further, a deployment of Ratatoskr would have to provide some degree of execution time predictability, suggesting that a release would use some other, more predictable sort of memory management, i.e. explicit memory allocation/deallocation.

### 5.3 Fault Models

GridStat uses multiple underlying network technologies, spans a wide area, and will sustain several usage patterns (usage patterns to this point includes rate based status updates and bursty RPC traffic). This makes for very complex behavior and it is difficult to provide a good fault model for GridStat without field testing. For this evaluation, two fault models were combined to account for the rich diversity of potential fault patterns in a GridStat deployment.

- *Omission-fault* - Each link is assigned a uniform probability of dropping each packet passing through it. The drop probability is the only variable for the omission-fault model. Each drop is completely isolated; no other link or later or earlier packet on the same link is affected by a drop. Omission-fault attempts to model temporally and spatially isolated drops in links where no retry-upon-failure is attempted below the transport layer in the protocol stack. Examples of uniform drop rate errors are background noise or very short term physical interference in links causing packet data corruption: passing physical objects in the way of the beam of a microwave beam, bursts of electromagnetic noise from power anomalies in a substation wired with copper or frequency noise from grid devices in a broadband over power link.
- *Duration-fault* - Each link is in one of two states: disabled or enabled. If disabled, all packets passing through the link are dropped. If enabled, the operation of the link is not affected, and all packets pass through the link unless omission failures occur. Links are ordinarily enabled, except for 1-second periods of disable state. Disabled state periods occur by a Poisson process, where the average number of occurrences per second ($\lambda$) is the only variable for the duration-fault model. Duration-faults model transient failures in network components. Examples may include: router maintenance, fiber cuts or local power-outs. Such failures may certainly have a duration of well over a second, but disabling for one second is enough to no-

tice the effects of duration failures on communication. It should be noted that dynamically routed networks will quickly adjust so signals will circumvent duration faults, even nearly instantaneously [12]. GridStat uses static routing for steady-state effciency and throughput, and so paths will not be adjusted even if faults are detected. This is future work, but the path calculations are costly and in many cases may not be worthwhile; this is why the primary mechanisms for overcoming longer-term failures of network links and status routers is spatial redundancy.

### 5.4 Experiment Procedure

The following experimental procedures were utilized:

- For all experiments, the process running the Ratatoskr client also functioned as experiment coordinator.
- For each experiment session, 10,000 RPC calls were made to warm up the system, and then one or more experiments were run sequentially.
- An experiment consists of 10,000 RPC calls with data gathered from each call.
- A new Ratatoskr connection was established for each experiment, and closed at the end of the experiment.
- All calls were made with an unlimited number of ACK/retries.
- Between each call and the next, all links emulated in the link emulator was reset, specifically by setting the internal clock used for duration failure $2 * (1/\lambda)$ seconds into the future, in effect ending all previous disabled states and allowing new ones to arrive.
- A link delay of 1 millisecond was used unless specified otherwise in the experiment description.
- When temporal redundancy was used, a two millisecond delay between redundant sends was used.
- For all experiments without spatial redundancy, calls were made over a path consisting of 6 emulated links. For experiments using spatial redundancy, calls were made over one path of 6 emulated links and one path of 7 emulated links.
- The timeout for the ACK/retry technique had a base of 25 milliseconds, allowing 12 ms for transfer delay, 20 ms for garbage collection and 3 ms for overhead from the link emulator, GridStat routing and Ratatoskr. Higher timeouts were assigned by Ratatoskr to sends with temporal or spatial redundancy due to the wait between redundant sends and the extra hop in the spatial paths. Specifically, using spatial redundancy added 2 ms to the timeout, and using temporal redundancy added 2 ms for each redundant send.

### 5.5 Result Data

The following data was extracted from each call:

- Time was measured from when the call was made until the result arrived (*end-to-end delay*).
- The number of timeouts experienced was recorded. This includes any timeouts the server experienced while attempting to send the result. Specifically, the number of timeouts for a call is the number of timeouts at the client before the first time a call arrives at the server *hence, not including timeouts introduced by missed ACKs)*, and the number of timeouts for the result to reach the client the first time (again not including timeouts introduced by lost ACKs).
- Early success rate for experiments is defined as the number of calls with no retries divided by the total number of calls.

### 5.6 Resiliency of Temporal Redundancy

To evaluate the resiliency of the temporal redundancy mechanism, a series of experimental runs was performed with increasing degrees of temporal redundancy (1, 2, 4 and 8 sends). Each of the temporal redundancy levels was tested over a set of omission fault rates (1%, 2%, 4% and 8%) applied to all emulated links. Duration loss was omitted from the evaluation, and is addressed in a later experiment. The results are shown in figure 4, with corresponding expected results from analysis. The analysis combines the drop rates of individual links into one drop rate for the 6-link path from client to server ($f_{6-link}$), and calculates the probability of at least one packet being delivered ($s_{send} = (1 - f_{6-link})^n$ for temporal redundancy level $n$. For the end-to-end loss probablility, the probabilities for successful delay of both the send and the return is $s^2_{send}$. The experimental results match the analysis very closely.

Figure 4 shows that two temporally redundant sends are not sufficient to entirely overcome a 1% loss rate, but with 99.2% early successes against 88.5% for the non-redundant calls it is still a good improvement. With 4 resends, 99.92% early successes are achieved at 4% failure rate. For 8 resends, 99.92% of calls were early successful even at 8% loss rate, where the end-to-end early success rate without reliability was 36.5%. Even during periods of intensive network loss, critical applications where the extra bandwidth for temporal redundancy can be spared should be able to

perform RPC calls with very few retries, given that the loss patterns accomodate temporal redundancy.

### 5.7 Resiliency of Spatial Redundancy

Spatial redundancy was evaluated in a manner similar to temporal redundancy. Runs were performed with spatial redundancy enabled over increasingly higher occurrence frequencies of duration faults (1 second failure every 10000 seconds, 1s/1000s, 1s/500s, 1s/100s and 1s/50s). Values for expected results based on analysis are included in the diagram. An analysis similar to the analysis for temporal redundancy was made, simplifying the arrival rates of fault durations for links as fault percentages (1s/50s=2%, 1s/100s=1%...).

For the experiments, 100% early successes was achieved for both 1s/10000s and 1/1000s fault rates. As this is for 10,000 calls, this is encouraging as it satisfies at least a 99.99% end-to-end reliability requirement. For 1s/500s fault durations, 99.96% early success is achieved.

### 5.8 Comparison to ACK/Retry-Only RPC

A final experiment compared the performance of RRPC to a traditional RPC call without other forms of reliability than ACK/retry, and evaluated the effect of spatial and temporal redundancy for the failure models used. Here, performance refers to the ability to tolerate network faults and end-to-end delay over a faulty network. Experiment runs for no redundancy, 4 temporally redundant sends, spatial redundancy and the combination of the two were performed over 1% omission fault rate, 1 second fault duration every 10000 seconds, the combination of the two and no failures. The early success rates are shown in figure 6. The average duration with standard deviation for all redundancy levels with both fault models combined is shown in figure 7. Cumulative distributions of timeouts experienced before call success for all redundancy levels for the combination of fault models is shown in figure 8 (note that the percentage ranges on the graphs are different to allow visible details). The highest calltimes measured with the combination of losses are: 939 milliseconds for no redundancy, 91 ms for spatial redundancy, 977 ms for temporal redundancy and 24 ms for the combination of the redundancy techniques.

From figure 6, it is seen that for 1% omission loss, temporal redundancy experiences very few timeouts, but the spatial redundancy does not provide enough reliability for the fault rate level and 0.81% of the calls experience timeouts. Without any redundancy, less than 90% of the calls achieve early success, which makes for very unstable calltimes. The 1s/10000s duration fault setting does not affect the early success rate of any of the redundancy levels too

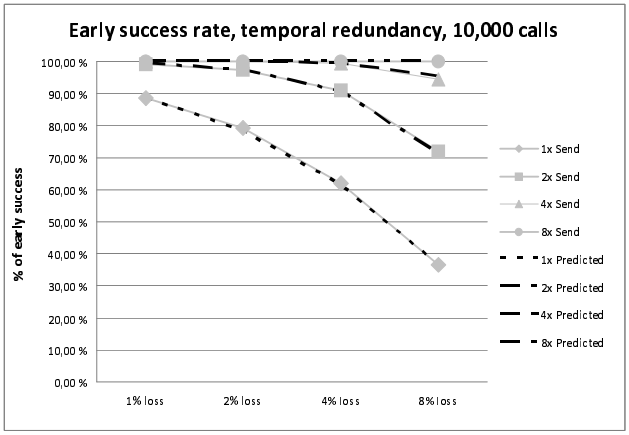


**Figure 4. Early success for temporal redundancy over varying omission fault rates**

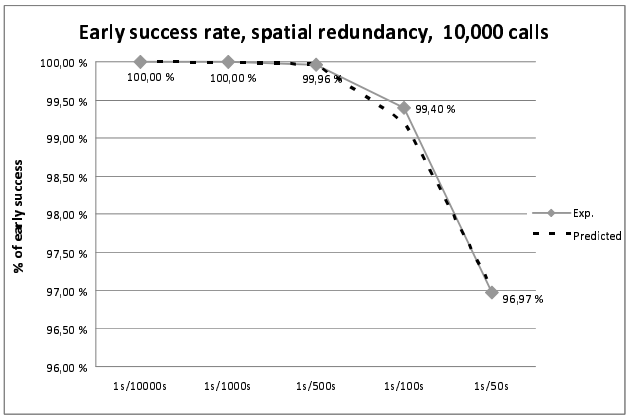


**Figure 5. Early success for spatial redundancy over varying duration faults**

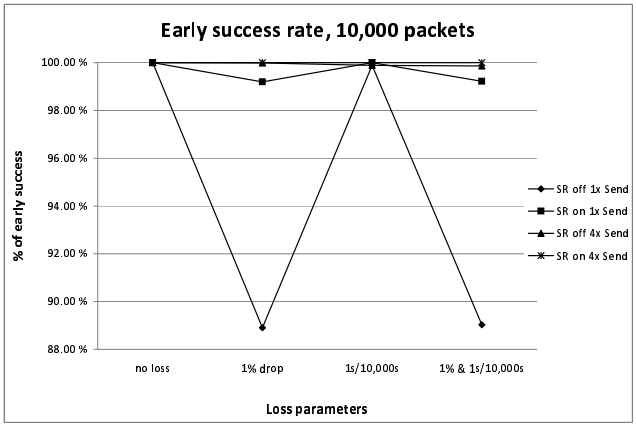


**Figure 6. Early Success for varying redundancy and loss**

much, while the combination of fault models has a pattern very similar to that of 1% omission failure.

In figure 7, the effects of duration loss becomes apparent. Temporal redundancy, which is ineffective against duration faults, has a considerably higher standard deviation of end-to-end delays than spatial redundancy. Together with the low early success rate, this signifies that a few calls must retry several times before success is achieved, even with four redundant sends. The experiment with no redundancy follows a similar pattern with high standard deviation of end-to-end delays, and also the average end-to-end delay is higher. This is from the high percentage of calls that timed out. The end-to-end delay with both forms of redundancy retain the same average as spatial and temporal redundancy, and with a small standard deviation. No loss was experienced during the sends with full redundancy. Figure 8 depicts standard deviation patterns. 99.85% all of the calls made with temporal redundancy incurred no timeouts, but the distribution has a long tail, and 0.05% of the calls incurred over 10 retries. The highest end-to-end delay measured for temporal redundancy was 977 milliseconds, and the highest number of retries was 22. Comparing the cumulative distribution for spatial redundancy to the temporal calls reveals that while a relatively high number of calls experience timeouts (over 0.8%), only a single call experiences the highest number of timeouts, 2, and 91 milliseconds was the highest measured end-to-end delay. With both redundancy techniques employed, no timeouts were experienced and the highest measured end-to-end delay was 24 milliseconds. Without redundancy, over 12% of the calls experienced timeouts and the highest number of timeouts was 25, with the measured end-to-end delay 939 milliseconds. This is a higher number of timeouts for a shorter measured end-to-end delay compared to the temporal redundancy call, as the timeout is set higher for temporal redundancy due to the 2 millisecond wait between redundant sends. From this we can conclude that even if a control mechanism over a single-path network uses a transport protocol with temporal redundancy, it may still experience very high call durations with longer-term failures in a single component on the path. Compared to RPC calls without the redundancy measures found in RRPC, spatial redundancy greatly lowers the worst-case end-to-end call duration, temporal redundancy improves the average call duration considerably, and the combination improves reliability greatly.

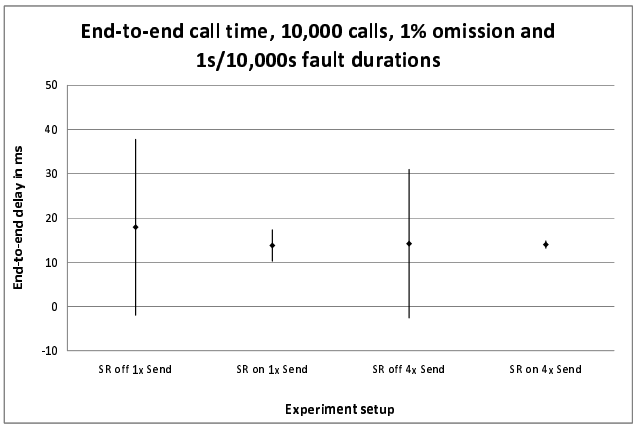


**Figure 7. Average calltimes for various redundancy with full loss**

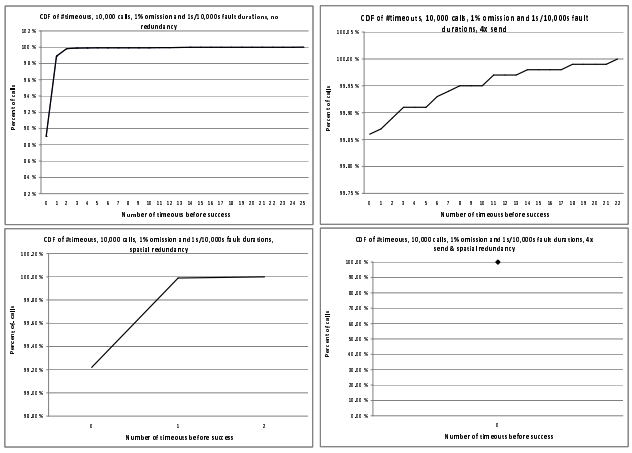

**Figure 8. Cumulative distributions of number of timeouts per call**

## 6 Related Work

Recent work on fault tolerant RPC mechanisms have been centered on host replication using distributed objects. As the distributed object interface is decoupled from the underlying implementation and environment, an object interface can be replicated into several implementations run-

ning in separate environments with minimum impact on observed behavior. Several CORBA implementations provide replicated objects, [16, 17, 13]. A replicated distributed object scheme, coupled with a real-time CORBA implementation, would provide timely delivery and fault-tolerance. While server replication mainly protects from server failure, protection from network failures would also be achieved to some degree. Depending on the redundancy scheme, sending to a group of replicated servers is achieved by multicasting from the client to the replicated hosts, and so the data is likely to go through several network paths. Such a scheme would still rely on the underlying network for network-level fault tolerance, and would not be able to reap the benefits of redundant path routing without modification. Further, object replication has to rely on strong multicast guarantees for synchronization between replicas to achieve correct group communication, which gives high worst-case messaging costs in the face of communication failures and thus scales badly with geographical distance.

We are currently aware of only one RPC mechanism utilizing temporal and spatial redundancy: [4, 15, 14] provides an implementation of CORBA for embedded systems providing hooks for spatial and temporal redundancy, but does not provide analysis or evaluation of the effects of these measures upon call reliability.

[8] is a study of the impact of path diversity on multihomed and overlay networks. They concluded that, on the best-effort Internet, a significant percentage of the paths from a multi-homed network might overlap, even when using multiple ISP. This is a strong argument against using the internet for critical infrastructures, where some key control and monitoring functionality requires very low latency and very high availability [5]. This points to the need for disjoint path management and a dedicated monitoring and control network for critical infrastructures.

## 7 Conclusions and Future Work

This paper presents the design, implementation, and evaluation of Ratatoskr, a novel RPC mechanism. Ratatoskr is split into two subsystems, a transport protocol for two-way communication over the GridStat network, the 2WoPS protocol; and an RPC mechanism built on top of the 2WoPS protocol, Ratatoskr RPC. The 2WoPS protocol utilizes the QoS semantics provided by the GridStat network to offer maximum delivery delay guarantees and spatially redundant network paths for sends. 2WoPS introduces two additional redundancy techniques: temporally redundant sends and ACK/retries. While the 2WoPS protocol was implemented specifically for the RPC mechanism, it is also used in another GridStat project that gained from the timeliness and reliability provided by the protocol. Ratatoskr RPC provides extensive customization of the redundancy levels for each call, providing a tradeoff space between timeliness, use of network resources, and reliability. Further, pre- and post conditions built into the call semantics provide additional safety mechanisms for application designers.

The evaluation explores the effectiveness of the redundancy techniques in the face of two fault-models, omission faults with uniform drop probability and durations of total link failure. The experiments show that both spatial and temporally redundant resends provide an exponential reduction in end-to-end fault occurrence rates for omission faults, with a a temporal redundancy of two providing a square reduction, a temporal redundancy of three providing cube reduction and so on. The experimental results match the expected results from the analysis. A comparison between Ratatoskr and a simulated traditional RPC mechanism without temporal and spatial redundancy, shows that Ratatoskr profits from the redundancy with a lower average for end-to-end call times and a considerably tighter call time distribution.

There are many future areas of research for Ratatoskr. Ratatoskr does currently not provide the security features required for a deployment in a critical infrastructure such as the power grid, and assumes no Byzantine behavior. Further, the prototype was implemented in Java, and the communication primitives does not accommodate platform independence. Finally, handling client and server failure is left for future work. This affects RPC failure semantics[19] in that only at-most once failure semantics are available. It should be noted that as the number of ACK/retries is customizable, the at-most once semantics found in Ratatoskr are more flexible than the traditional understanding, i.e. maybe once semantics may be achieved by setting the number of retries for a call to zero. If at-least once semantics or some form of exactly-once semantics should be implemented for Ratatoskr, similar flexibility would arise. Further information on the properties of Ratatoskr fault-semantics may be found in [20].

## Acknowledgements

We thank Erik Solum for his advice on this research; Jim Kusznir for advice and technical help with the cluster; and Dave Anderson, Ginny Hauser, and Loren Hoffman for their careful proofreading of this paper. This research has been supported in part by Grants CNS 05-24695 (CT-CS: Trustworthy Cyber Infrastructure for the Power Grid(TCIP)) and CCR-0326006 from the US National Science Foundation.

## References

[1] S. F. Abelsen. Adaptive gridstat information flow mechanisms and management for power grid contingencies. Mas-

ter's thesis, Washington State University, Pullman, Washington, USA, August 2007.

[2] D. Bakken, T. Evje, and A. Bose. Survivable status dissemination in the electric power grid. In *Proceedings of the Information/System Survivability Workshop (in Supplemental Proceedings of DSN2001)*, July 2001.

[3] D. E. Bakken, C. H. Hauser, H. Gjermundrød, and A. Bose. Towards more flexible and robust data delivery for monitoring and control of the electric power grid. Technical Report TR-GS-009, School of Electrical Engineering and Computer Science, Washington State University, May 2007. Available at http://www.gridstat.net/publications/TR-GS-009.pdf.

[4] K. Dorow and D. Bakken. Flexible fault tolerance in configurable middleware for embedded systems. In *Proceedings of the Workshop on Architectures for Complex Application Integration (WACAI2003)*. IEEE, 2003.

[5] EPRI/CEIDS. The integrated energy and communication systems architecture (IntelliGrid), volms i–iv. http://www.epri.com/IntelliGrid/, July 2004.

[6] K. H. Gjermundrød. *Flexible QoS-managed status dissemination middleware framework for the electric power grid*. PhD thesis, Washington State University, Pullman, Washington, USA, August 2006. Available at http://www.dissertations.wsu.edu/Dissertations/Summer2006/k%5Fgjermundrod%5F072406.pdf.

[7] K. H. Gjermundrød, D. E. Bakken, C. H. Hauser, and A. Bose. GridStat: A flexible QoS-managed data dissemination framework for the power grid. *IEEE Transactions on Power Delivery Systems*, 2008. To appear. Available via http://www.gridstat.net.

[8] J. Han and F. Jahanian. Impact of path diversity on multihomed and overlay networks. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*. IEEE, June 2004.

[9] C. Hauser, D. E. Bakken, and A. Bose. A failure to communicate: next generation communication requirements, technologies, and architecture for the electric power grid. *Power and Energy Magazine, IEEE*, 3:47–55, March-April 2005.

[10] V. Irava. *Low-cost delay-constrained multicast routing heuristcs and their evaluation*. PhD thesis, Washington State University, 2006. Available via http://www.dissertations.wsu.edu/Dissertations/Summer2006/v%5Firava%5F072106.pdf.

[11] R. Johnston. Obtaining high performance phasor measurements in a geographically distributed status dissemination network. Master's thesis, Washington State University, 2005. Available at http://www.dissertations.wsu.edu/Thesis/Summer2005/r%5Fjohnston%5F072905.pdf.

[12] S. Lee, Y. Yu, S. Nelakuditi, Z.-L. Zhang, and C.-N. Chuah. Proactive vs reactive approaches to failure resilient routing. *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM*, 1:176–185, 2004.

[13] S. Maffeis. Adding group communication and fault-tolerance to CORBA. In *COOTS'95: Proceedings of the USENIX Conference on Object-Oriented Technologies on USENIX Conference on Object-Oriented Technologies (COOTS)*, Berkeley, CA, USA, 1995. USENIX Association.

[14] A. D. McKinnon. *Supporting fine-grained configurability with multiple quality of service properties in middleware for embedded systems*. PhD thesis, Washington State University, Pullman, Washington, USA, December 2003. Available at http://www.dissertations.wsu.edu/Dissertations/Fall2003/a_mckinnon_093003.pdf.

[15] A. D. McKinnon, D. E. Bakken, and J. C. Shovic. A configurable cryptography subsystem in a middleware framework for embedded systems. *Computer Networks, Elsevier*, 46(6):771–795, 2004.

[16] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998.

[17] Y. J. Ren, D. E. Bakken, T. Courtney, M. Cukier, D. A. Karr, P. Rubel, C. Sabnis, W. H. Sanders, R. E. Schantz, and M. Seri. AQuA: An adaptive architecture that provides dependable distributed objects. *IEEE Trans. Comput.*, 52(1):31–50, 2003.

[18] D. C. Schmidt, D. Levine, and S. Mungee. The design and performance of real-time object request brokers. *Computer Communications*, 21, 1998.

[19] A. Z. Spector. Performing remote operations efficiently on a local computer network. *Commun. ACM*, 25(4):246–260, 1982.

[20] E. S. Viddal. Ratatoskr: Wide-area actuator RPC over GridStat with timeliness, redundancy, and safety. Master's thesis, Washington State University, Pullman, Washington, USA, December 2007.

[21] WSU. GridStat Avista demo. http://gridstat.net/avista.php.

[22] J. Zinky, D. Bakken, and R. Schantz. Architectural support for quality of service for CORBA objects. *Theory and Practice of Object Systems*, 3(1):55–73, April 1997. Available at http://www.dist-systems.bbn.com/papers/1997/TAPOS/.