The Architecture of the Starfish System: Mapping the Survivability Space

Kim Potter Kihlstrom  
Department of Mathematics and Computer Science  
Westmont College  
955 La Paz Road, Santa Barbara, CA 93108  
email: kimkihls@westmont.edu

Priya Narasimhan  
Department of Electrical and Computer Engineering  
Carnegie Mellon University  
5000 Forbes Avenue, Pittsburgh, PA 15213-3890  
email: priya@cs.cmu.edu

Christopher Phillips, Christopher Ritchey, and Benjamin LaBarbera  
Department of Mathematics and Computer Science  
Westmont College, 955 La Paz Road, Santa Barbara, CA 93108  
email: {chphilli,critchey,blabarbe}@westmont.edu

ABSTRACT
Starfish is a new system that provides intrusion detection and intrusion tolerance for middleware applications operating in an asynchronous distributed system. The Starfish system contains a central, highly secure and tightly coupled “body.” This body is augmented by “arms” that are less tightly coupled and that have less stringent security guarantees, each of which can be removed from the body if a significant security breach occurs. New arms can be “grown” as needed. Residing between the body and arms are “shoulders” that have intermediate guarantees.

The Starfish system aims to employ a number of techniques for providing proactive survivability, allowing the system to provide critical services even after the occurrence of attacks, accidents, or faults. Starfish is aimed at supporting distributed applications such as Web Services. The specific contributions that we make in this paper are to present dimensions in the survivability space, to provide a mapping of a number of prior systems to the survivability space, and to give a mapping of the three regions in Starfish to that space. We describe the architecture of the Starfish system, and identify specific mechanisms present in each of the regions of Starfish.

KEY WORDS
survivability, middleware, object-oriented systems, fault-tolerant systems, security

1 Introduction
The Starfish system [1] is a new system currently under development that aims to provide proactive survivability to middleware applications, allowing them to continue to operate correctly and reliably despite intrusions, malicious attacks, faults, and accidents that may corrupt the processors, communication links, or objects that comprise the system.

The Starfish system contains three regions as shown in Figure 1, each with its own security guarantees, fault model, mechanisms, and protocols. Nodes in the body, or core, operate under stringent security requirements and a Byzantine fault model to provide service guarantees for critical applications that must be protected from malicious attack. Nodes in an arm assume less stringent security requirements and crash/omission failure semantics to provide service guarantees for applications that are non-critical and not anticipated to be subject to malicious attack. In the event of corruption, an arm can be removed from the system. In addition to the body and arms, Starfish contains intermediate regions, which we refer to as shoulders. Nodes in a shoulder have intermediate service requirements and survivability guarantees.

2 Underlying Model
We consider an asynchronous distributed system consisting of processors that communicate by sending messages over a network. Processors have access to local clocks, but these clocks are not synchronized.

---

1Starfish are known to have small bodies, out of which spring forth a varying number of arms, which break off when damaged. These arms subsequently heal and re-grow.
The system is subject to communication, processor, and object faults. Communication between processors is unreliable, and thus messages may need to be retransmitted and may be arbitrarily delayed. Communication channels are not assumed to be FIFO or authenticated, and messages may be corrupted in transit. The network may partition, leaving some nodes disconnected from other nodes.

Processors are either correct or may suffer from a variety of faults. A correct processor always behaves according to its specification. A crashed processor ceases to send any messages or to perform any processing. A crashed processor can recover and start up again, in which case it retains no information regarding its state prior to the crash. A processor may also suffer from an omission fault, in which case the processor fails to respond to a sequence of events or fails to send a sequence of messages. A processor may suffer from a timing fault, in which case it responds to an event or sends a message too early or too late.

A Byzantine processor, also referred to as a malicious processor, exhibits arbitrary behavior. A Byzantine processor may attempt to disrupt the system by sending different information to different processors, purporting that it is the same information, or by selectively sending information to some processors and not to others. Further, a malicious processor can attempt to thwart the consistent ordering of operations by sending messages to different processors in a different order. Such a processor can also masquerade as another processor by including the identifier of another processor in the messages it sends. A Byzantine processor can refuse to send messages, can send messages that are syntactically or semantically incorrect, or can claim that another processor is faulty.

Objects also may be correct or subject to faults. A correct object always behaves according to its specification. A crashed object ceases to perform any processing or to send any invocations or responses. A crashed object may recover and start up again. A faulty object may fail to send an invocation or response as required by the application, which we refer to as an omission fault. An object may suffer from a timing fault, in which case it sends an invocation or response too early or too late. A faulty object may also send a message that contains an incorrect (corrupted) invocation or response, which we refer to as a value fault.

A Byzantine object may attempt to disrupt the system by sending different invocations or responses to different objects, purporting that it is the same information, or by selectively sending invocations or responses to some objects and not to others. Further, a Byzantine object can attempt to thwart the consistent ordering of operations by sending invocations or responses to different objects in a different order. Such an object can also masquerade as another processor by including the identifier of another object in the messages it sends. A Byzantine object can refuse to send messages, can send messages that are syntactically or semantically incorrect, or can claim that another object is faulty.

3 Survivability Space

In seeking to design survivable systems, we must first define what we mean by survivability. Survivability includes and goes beyond reliability and security. Reliability encompasses the system’s ability to tolerate faults, while continuing to provide useful services. Security involves protecting the resources of the system against malicious attack, and ensuring the confidentiality and integrity of the application data. Survivability goes beyond traditional computer security and reliability in that survivable systems must continue to provide critical services even after the occurrence of attacks, accidents, or faults. Thus, in describing survivability, three dimensions suggest themselves: reliability, security, and quality of service provided. These three dimensions are shown in Figure 2. The origin of the survivability space represents a system that provides none of the desired survivability properties. As one moves along the security axis, the level of security provided by the system increases, and similarly for the reliability and quality of service axes. A particular point in the three dimensional space would represent a particular combination of the three aspects of security, reliability, and quality of service.

While the three dimensional survivability space is appealing in its simplicity, it should be apparent that each of these three dimensions themselves encompass multiple aspects. For example, security includes confidentiality as well as data integrity. Reliability encompasses tolerance to processor faults as well as communication faults, and quality of service includes performance as well as scalability. We thus make use of a successively lower level of abstraction to view each individual axis as a three dimensional subspace in itself, as shown in Figure 3.

3.1 Reliability Subspace

In defining the reliability subspace we focus on the fault tolerance of the system. Three types of faults suggest themselves as three dimensions of the reliability subspace: processor faults, communication faults, and object faults. Greater distances from the origin imply tolerance to more severe fault models.
Confidentiality may be provided by encryption of messages and files. Confidentiality may also be provided by using a threshold scheme, in which files or objects are divided into shares which are then distributed across multiple hosts. The information can only be reconstructed if enough shares are available, so that if a single host is compromised, an attacker is still not able to obtain the complete file or object.

Various methods exist which provide authentication. Weakest of all is simply to use a unique processor or user identifier. This may be strengthened through the use of passwords. Message authentication codes allow for a hash function to be combined with a key for greater authentication. Digital signatures, which also involve the use of keys, result in even stronger authentication guarantees.

The integrity of the system can be enhanced through the detection of faulty or corrupted nodes and the removal of such nodes. Integrity also includes the recovery of the system from faults or corruption, which may make use of log or history files to perform system rollback, or may provide transfer of state information to a new or recovered node. Integrity may be achieved through proactive techniques involving containment and prevention of infection or corruption.

### 3.3 Quality of Service Subspace

Various dimensions could be chosen for the quality of service subspace. We choose to focus on adaptivity, scalability, and performance.

Adaptivity encompasses the ability of the system to respond and adapt to changes in its current status such as load and connectivity, as well as varying requirements of the application. It includes the ability to remove and add processors, applications, objects, and object replicas. It also may include the ability to migrate nodes from one area of the system to another, or to migrate object replicas from one processor to another. Adaptivity may also include live software upgrades or evolution.

Scalability includes the ability to scale with the number of processors, the number of objects, and the number of applications. It also involves the ability to scale from a local area network to a wide area network or internet. Greater distance from the origin indicates larger or wider scale.

Performance includes metrics such as latency and throughput. As one moves along the performance axis, a system providing higher throughput and lower latency is represented.

### 4 Mapping Prior Systems

In Figures 4 and 5 we show mappings to the survivability space of thirteen prior systems, as described below.

Starfish is built upon the authors’ prior experience with the Immune system [3]. The Immune system (Figure 4) provides survivability to CORBA applications trans-
Figure 4. Mapping prior systems to the survivability subspaces.
parently, enabling them to continue to provide useful services despite processor crash, omission, timing, and Byzantine faults as well as object replica crash, omission, timing and Byzantine faults. It also handles communication faults including message loss, delay, and corruption, but does not handle network partitioning. With regard to security, and specifically system integrity, Immune provides for detection, removal, and recovery from faulty processors and objects, but does not provide any restriction or prevention. Immune allows for the use of unique identifiers and digital signatures for authentication, but makes no provision for confidentiality. In the quality of service subspace, Immune does not scale well in the number of processors, applications, or objects, and is designed only for a local area network. Immune has much higher latency and lower throughput than systems that are designed without provision for security or tolerance to Byzantine faults or malicious corruption.

The Immune system was the result of integrating the SecureRing system and the Eternal system. The SecureRing system [4] provides secure reliable totally-ordered message delivery as well as group membership services even when subject to Byzantine processor faults. Messages are multicast to processor groups and delivered in a consistent total order to all members in the group. The Eternal system [5] (Figure 4) is intended to provide fault tolerance as well as evolution to CORBA-based applications. Replicas of the object are distributed throughout the system, with consistency being maintained by using the underlying group communication protocols for communication between objects. Eternal provides reliability despite processor and object replica crash, omission, and timing faults as well as communication faults including message delay, message loss, and network partitioning faults. It does not provide tolerance to Byzantine processor or object faults, and does not handle message corruption. In the area of security, Eternal provides for detection and removal of, as well as recovery from, processor and object faults, but does not provide for restriction or prevention. It provides only rudimentary authentication in the form of processor identifiers, and does not provide any confidentiality. With regard to quality of service, Eternal provides detection and removal of crashed objects and processors as well as live software upgrades. It achieves high throughput and low latency, but does not scale well in the number of processors, applications, or objects. Eternal is designed for a local area network and does not scale to a wide area network or internet.

The MAFTIA middleware architecture [6] (Figure 4) supports multi-party interactions reliably under a group oriented middleware suite. Designed to run in a potentially hostile environment, vulnerabilities, attacks, and intrusions are targeted for support. MAFTIA makes use of an unreliable fault detector to deal with possible attacks. It tolerates Byzantine object and processor faults, and handles message loss, delay and corruption, but not network partitioning. It provides digital signatures for authentication, and encryption for confidentiality. It makes use of detection, removal, and recovery, and also supports prevention. MAFTIA uses a network system consisting of LANs situated on a WAN to support wide-area stability, but does not scale well in the number of processors, objects, or replicas. It achieves good performance for a Byzantine tolerant system.

The Secure Spread system [7] (Figure 4) provides for reliable, secure group communication in either local area or wide area networks. Security is provided by Secure Spread’s modular drop-in design of encryption and/or key agreement protocols. It tolerates processor and object crash, timing, and omission faults, but not Byzantine faults. Secure Spread handles message loss, delay, and corruption as well as network partitioning. It provides authentication as well as detection, removal, and recovery. It operates in a wide or local area network, scales well in the number of processors, applications, and objects, and allows for removal and addition, but does not achieve high performance.

The Phalanx software system [8] (Figure 4) stores shared data abstractions such as files and locks in a persistent object store. It provides tolerance to Byzantine processor and object faults, and handles communication faults such as message loss, delay, and corruption, but not network partitioning. With regard to security, Phalanx uses identifiers and digital signatures for authentication, and provides for detection and removal of faulty nodes. It does not provide for encryption of messages or a threshold scheme. The system is scalable to a large number of servers and a wide area network or internet. The performance is poor compared to systems that do not tolerate Byzantine faults.

The Fleet middleware system [9] (Figure 4) is a generalization of the Phalanx system, allowing for persistent objects to be of arbitrary types, techniques for fault monitoring, and dynamic restructuring of the infrastructure using new object types. Objects controlled by the Fleet system are maintained in a persistent manner that allows them to outlive the clients that originally created them. The replication process allows for malicious corruption of servers, while still maintaining integrity. The Fleet system was designed for use in Java/Jini applications, and makes use of Byzantine quorum systems, enabling scalability. The mapping of Fleet to the survivability space is the same as that of Phalanx.

The AQuA architecture [10] (Figure 4) tolerates crash, timing, and omission faults among processors and objects. It does not protect against Byzantine processor faults, but does provide for protection from value faults due to corrupted objects within a CORBA-based dependability framework. AQuA is not designed to detect or protect against message corruption faults, but does handle message loss and delay. With regard to security, AQuA uses identifiers and provides for integrity through detection, removal, and recovery, but does not provide confidentiality. AQuA does not scale well in the number of processors, applications, or objects, and is designed for a LAN. It allows for removal and addition, and achieves high performance.
Figure 5. Mapping prior systems to the survivability subspaces.
ITDOS [11] (Figure 5) is an intrusion tolerant CORBA system that provides high availability and high integrity in the face of attacks or intrusions. It makes use of a Byzantine fault tolerant multicast protocol and employs majority voting. ITDOS allows for heterogeneous application implementations. It provides tolerance to Byzantine processor and object faults, as well as communication faults including message loss, delay, and corruption, but does not handle network partitioning. It provides authentication including MACs, and uses encryption to provide confidentiality. Integrity is attained through detection and removal of faulty nodes. It scales well in the number of objects, but not in the number of applications or processors, and is designed for a local area network. The performance is low compared to systems that are less secure or that do not provide Byzantine fault tolerance.

The Object Group Service (OGS) [12] (Figure 5) provides a CORBA-compliant approach to fault tolerance in CORBA applications utilizing a set of CORBA services. It tolerates processor and object crash, timing, and omission faults, but does not handle Byzantine processor or object faults. OGS handles message loss and delay as well as network partitioning, but does not handle message corruption. With regard to security, OGS employs identifiers and makes use of detection, but does not provide any confidentiality. It does not scale well in the number of objects, applications, or processors, and achieves low performance.

The InterGroup group communication suite [13] (Figure 5) is designed to scale to a large number of nodes and to wide-area networks and internets. There is also a secure group layer that provides security for groups, including authentication and encryption. InterGroup provides tolerance to processor and object crash, timing, and omission faults, but not Byzantine faults. It handles network partitions and message loss and delay, but not message corruption. It provides for detection, removal, and recovery. InterGroup allows for adaptivity in removal and addition, and achieves medium performance.

The BFT library [14] (Figure 5) is a state machine replication system that is tolerant of Byzantine processor and object faults. The system tolerates message loss, delay, and corruption, but not network partitioning. BFT allows for authentication including MACs and digital signatures; high performance is possible due to digital signatures only being implemented in regard to view-change and new-view messages, while most messages are authenticated using only message authentication codes (MACs). BFT does not provide for confidentiality. With regard to integrity, BFT performs detection, removal and recovery. Prevention is provided in the form of proactive recovery of replicas, allowing a system to tolerate any number of faults within the system as long as at least 2/3 of the replicas in the system are clean in any given time. Mechanisms are also provided for system state transfer. BFT does not scale well in the number of objects, applications, or processors, and is designed for a local area. It provides adaptivity in allowing for removal and addition, and achieves good performance for a Byzantine tolerant system.

The FRIENDS system [15] (Figure 5) provides the capability to build fault-tolerant applications through the use of libraries of metaobjects. Fault tolerance, security, and group communication can be provided by making use of separate metaobjects. FRIENDS provides tolerance to processor and object crash, omission, and timing faults, but does not provide Byzantine object or processor tolerance. It handles message loss and delay, but not message corruption or network partitioning. Several subsystems are included in the composition of the FRIENDS system, including a subsystem providing support for replication and detection of faults, and a subsystem providing secure communication through authentication, authorization and audits. Encryption provides confidentiality, and detection, removal, and recovery is supported. The system does not make use of majority voting; it instead uses digitally signed messages that are assumed to be from the client that is claiming the message. FRIENDS does not scale well and is designed for a local area network. It allows for removal and addition, and achieves medium performance.

The Cactus project [16] (Figure 5) is a framework that makes use of “micro-protocols,” which each implement a specific property in a flexible system of survivability. One micro-protocol supports active and passive replication to provide fault tolerance, while another provides security utilizing various authentication and encryption methods. The middleware service GroupRPC allows tolerance of crash, omission, timing, and Byzantine processor and object faults by utilizing the Cactus approach. Cactus allows tolerance to message loss, delay, and corruption, but does not provide for network partitioning. It is designed for a WAN, but does not scale well in the number of objects, applications, or processors. It achieves medium performance.

5 The Architecture of Starfish

As shown in Section 4, tradeoffs exist between the performance attained, the level of reliability, and the security guarantees. Systems that provide Byzantine fault tolerance, for example, achieve lower performance than systems that protect against only crash faults. Systems that provide fault tolerance do so by replicating and distributing data, which inherently makes the data more vulnerable to a security breach.

Any given enterprise will have different requirements for different applications with regard to security, reliability, and quality of service, and one system will typically not fit all of the needs of the enterprise. In Starfish, we aim to make different tradeoffs available in the same system, by providing the body, shoulder, and arm regions. The “crown jewels” of the enterprise, those applications and data that must remain highly secure and reliable, will reside in the body, while less critical applications and data can be located in the shoulders or arms. Thus, support is provided
for multi-tiered systems with varying guarantees for survivability.

Starfish is able to provide scalable wide-area support to connected enterprises such as Web services. Multiple Starfish systems can be attached using their arms, as shown in Figure 6. From the view of one enterprise, a linked Starfish always appears as an arm.

In Figure 7 we show a mapping to the survivability subspaces of the body, shoulder, and arm regions of the Starfish system, with the body shown in light, shoulders in medium, and arms in dark color. As shown in Figure 7, the body provides higher fault tolerance and security guarantees, while correspondingly achieving a lower quality of service, than that of the shoulders and arms. As we move outward from the body, a higher quality of service is attained, at the cost of lower fault tolerance and security.

The architecture of the Starfish system is shown in Figure 8. The interceptor intercepts application messages that are intended for TCP/IP and instead passes them to the replication manager. The replication manager is responsible for replicating objects and distributing those objects across the system, performing majority voting on both invocations and responses, and communicating invocations and responses to all replicas of the target object through the group communication protocol. The recovery manager is responsible for state transfer and rollback operations when necessary, and maintains log files. The group communication protocols include layers for internal communication within the system, group communication involving hierarchical groups, and external communication between processors. Additional layers provide security and fault detection.

The survivability guarantees of each of the regions of the Starfish system, along with some of the mechanisms used to achieve those guarantees, are summarized in Table 1 and described in Sections 5.1 to 5.3 below.

---

5.1 Body

In the body region of Starfish, tolerance is provided to processor and object timing, crash, omission, and Byzantine faults. Communication faults including message loss, delay, and corruption are handled, but the network is assumed not to partition. With regard to security, the body is designed to provide high levels of confidentiality, authentication, and integrity. However, the price paid for these high reliability and security guarantees is in the quality of service attained. The throughput will be low, and the latency high, compared with systems that do not provide Byzantine fault tolerance or security. Similarly, the body region does not scale well in the number of processors, applications, or objects, and is designed only for a local area network. Only a medium level of adaptivity is provided.

The specific mechanisms that provide the reliability properties include active replication and majority voting. When combined with a secure reliable ordered multicast protocol, these techniques ensure that all correct object
<table>
<thead>
<tr>
<th>Survivability Aspects</th>
<th>Node Types</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Body</td>
</tr>
<tr>
<td>Reliability Fault Model</td>
<td>processor timing, crash, omission, and Byzantine replica timing, crash, omission, and Byzantine message loss, delay, and corruption</td>
</tr>
<tr>
<td>Mechanisms</td>
<td>active replication</td>
</tr>
<tr>
<td></td>
<td>majority voting</td>
</tr>
<tr>
<td></td>
<td>secure reliable ordered multicast</td>
</tr>
<tr>
<td></td>
<td>Byzantine value fault detector</td>
</tr>
<tr>
<td>Security Guarantees</td>
<td>high confidentiality</td>
</tr>
<tr>
<td></td>
<td>high authentication</td>
</tr>
<tr>
<td></td>
<td>high integrity</td>
</tr>
<tr>
<td>Mechanisms</td>
<td>threshold scheme encryption</td>
</tr>
<tr>
<td></td>
<td>identifiers</td>
</tr>
<tr>
<td></td>
<td>passwords</td>
</tr>
<tr>
<td></td>
<td>signatures</td>
</tr>
<tr>
<td></td>
<td>Byzantine fault detector</td>
</tr>
<tr>
<td></td>
<td>value fault detector</td>
</tr>
<tr>
<td></td>
<td>secure membership</td>
</tr>
<tr>
<td></td>
<td>logging</td>
</tr>
<tr>
<td></td>
<td>state transfer</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality of Service Guarantees</td>
<td>low throughput</td>
</tr>
<tr>
<td></td>
<td>high latency</td>
</tr>
<tr>
<td></td>
<td>low scalability</td>
</tr>
<tr>
<td></td>
<td>medium adaptivity</td>
</tr>
<tr>
<td>Mechanisms</td>
<td>removal/addition migration</td>
</tr>
<tr>
<td></td>
<td>hierarchical groups message prioritization optimistic delivery</td>
</tr>
<tr>
<td></td>
<td>removal/addition migration</td>
</tr>
<tr>
<td></td>
<td>hierarchical groups message prioritization optimistic delivery</td>
</tr>
</tbody>
</table>

Table 1. Survivability aspects for different node locations.
replicas process the same set of operations in the same order, and thus remain consistent despite malicious replicas that may attempt to cause inconsistency. While a malicious replica may provide incorrect invocations or responses, the majority voting ensures that clients process only correct invocations or responses, and the value fault detector enables a malicious replica to be detected and removed from the object group membership. At the processor level, a Byzantine fault detector [17] enables the detection of a malicious processor, and the processor membership algorithm is then able to remove it. Vaccination techniques serve to train the system based on fault injection.

With respect to security, confidentiality is achieved through encryption and through a $p$-$m$-$n$ threshold scheme, in which information is broken into $n$ shares such that any $m$ of the shareholders can reconstruct the information and a group of fewer than $p$ shareholders gains no information. Unique identifiers, passwords, MACs, and signatures are used to achieve authentication. With regard to integrity, the specific mechanisms include a Byzantine fault detector, value fault detector, secure membership protocol, logging, anomaly detection, sandboxing, intrusion history, message digests, state transfer, quarantine, and vaccination.

The mechanisms that contribute toward quality of service include removal and addition of nodes as well as object migration, all of which enhance adaptivity. Hierarchical groups [1] improve scalability, while mechanisms to achieve lower latency and higher throughput include message prioritization and optimistic delivery.

5.2 Shoulders

In the shoulders region, tolerance to object replica and processor timing, crash, and omission faults is achieved, but not Byzantine fault tolerance. Communication faults such as network partitioning as well as message loss, delay, and corruption are handled. In the area of security, no confidentiality is provided, but medium levels of authentication and integrity are achieved. With regard to quality of service, medium throughput, medium latency, medium scalability, and high adaptivity are provided.

The specific mechanisms that are used to achieve reliability include passive replication combined with reliable ordered multicast, a value fault detector, and group membership. Authentication is provided through the use of unique identifiers, passwords, and MACs. Integrity is attained through the value fault detector, membership, message digests, logging, and state transfer. Adaptivity is provided through live upgrades, node removal and addition, and object migration. Hierarchical groups improve scalability, while performance is enhanced through message prioritization and optimistic delivery.

5.3 Arms

In the arms, the specific faults tolerated are object replica and processor timing, crash, and omission. Communication faults including message loss and delay, as well as network partitioning, are also handled. In the area of security, no confidentiality, only a low level of authentication, and medium integrity properties are provided. In contrast to the body, the quality of service provided is much higher in terms of throughput, latency, and adaptivity. The arms provide scalability in the number of applications, objects, and processors, and is designed for a wide area network or internet.

The specific mechanisms that provide reliability properties include timeouts and retransmission. Authentication is provided through unique identifiers and passwords. Integrity is attained via detection, logging, state transfer, removal, and recovery techniques. With regard to quality of service, adaptivity is provided through live upgrades, removal and addition of nodes, and object migration. Scalability is increased through the use of hierarchical groups, and performance is enhanced through message prioritization and optimistic delivery.

6 Conclusions

We have presented a mapping of the survivability space, identifying the three major axes as security, reliability, and quality of service. We have also shown that each of these axes is itself another space, and we have identified components of each of these subspaces. We have looked at thirteen prior systems and shown the mapping to the survivability space of each of these systems.

We have introduced the architecture of the Starfish system, and have described the mapping to the survivability space of the body, shoulders, and arms of Starfish. We
have also identified a number of specific mechanisms that will enable Starfish to achieve the identified properties.

The Starfish system is currently under development. Future work will include implementing, testing, and obtaining performance data for the system, as well as the development of additional mechanisms to attain high quality of service while achieving desired reliability and security guarantees.

References


