Design Elements for a Space Information Network Operating System

Anders Fongen
Norwegian Defence University College (FHS)
Lillehammer, Norway
email: anders@fongen.no

***Abstract*—Space Information Networks (SIN) have quite different characteristics from ordinary distributed systems and clouds. Therefore, the middleware and operating systems governing the service provision in SIN spacecrafts must manage the resources involved, as well as the lifecycle of the components that depend on these resources. This paper goes into detail in the distinguishing characteristics and proposes a blueprint for software design.**

***Keywords*—LEO satellites; space information networks; distributed OS; mobile computing.**

I. INTRODUCTION

The term *Space Information Network* (SIN) describes a set of satellites that cooperatively offer services for information processing and sharing, as well as traditional communication services. SIN is regarded as a natural evolution of satellite services, from radio mirrors in geostationary orbit and Low Earth Orbit (LEO) constellation for communication services (e.g., Iridium) [1][2]. Among expected benefits from a SIN is (1) very low end-to-end latency, as low as 3 ms, and (2) global coverage. A SIN is likely to drive new applications which require these properties.

In a series of previous publications, different aspects of SIN operation (architecture [3], security [4], cache management [5], routing [6], session state management [7]) and data sharing [8] have been addressed. Building on these studies, this position paper proposes a design blueprint of a middleware/operating systems, which offers the necessary services and defines Application Programming Interfaces (APIs) to ground based clients, service components and service containers operating in each spacecraft. Within the presented article, the term “Space Information Network Operating System” will be abbreviated “SIN-OS”.

The perspective of the presented analysis is that of Distributed Computing. Technical and physical properties of satellites related to energy management, antenna design, modulation, coding, jamming resistance etc., are not taken into consideration.

The remainder of the article is organized as follows: In Section II, some of the characteristics of SIN operations are identified as premises for the analysis, and Section III presents the components and services of the proposed SIN-OS design. The specific details of the proposed API are discussed in Section IV. For future study, the software simulator to be used is briefly presented in Section V, and the article draws its conclusion in Section VI.

II. OVERARCHING DESIGN CONSIDERATIONS

Central to the efforts presented in this manuscript are architectural properties, which heavily influences the software design. A selection of these properties are listen in the following paragraphs:

A. The N-layer Structure

The N-layer structure of service producers and service consumers is a typical property of any distributed systems, which also applies to a SIN. Any entity which offers services to a client may be a client to a service at a “deeper” level, and these relations form a tree structure rooted in the spacecraft, which connects to the surface client.

Some rules are chosen to simplify the design slightly:

- A surface client can only access one single service endpoint, and therefore connects to a single tree of service providers.
- There is a distinction between *servers* and *clients* at the surface. A space client can access a surface service, but not the other way around. Surface clients will never receive service requests.

B. Handover Operations

Surface based clients are stationary, while the orbiting elements are not, which causes a series of handover operations to take place for the sake of link maintenance. While the link budget for inter-satellite links to some extent can be estimated, the link from a surface client is dependent on nearby buildings and terrain and cannot be easily calculated in advance. The general problem of handover, whether between spacecrafts or from a surface client, are approached with the following rules:

- A handover operation is always initiated from the client side
- A handover operation is prepared and conducted by the server side, subsequent to a client request.

The following steps indicate the necessary actions taken during a handover operation: (1) The client notifies the server that a handover is requested, (2) The server decides which server is the best candidate to take over, and (3) transfer the session state to this candidate, then (4) inform the client of the new endpoint to use. Finally, (5) the client establishes a connection to the new server as indicated in step (4) and resumes the dialogue.

Handover is a solved problem in satellite constellations that offer stateless communication services, e.g., Iridium. This manuscript will therefore focus on problems related to handover operations where *stateful* and *collaborate* services are offered. In that case, handover also involves the migration of all data that constitutes the session state.

A handover request are likely to create cascades of new handover requests propagating through the tree of service providers. Client-server relations between spacecrafts are assumed to take place between units orbiting in the same direction, which may cause handover operations in one link to initiate handovers in the next link of the service chain, in the worst case, through all the orbiting units involved in the service provision.

C. Stateful Migration

A stateful service component in a spacecraft needs to migrate its units of execution during a handover operation. Stateless services are easily migrated if migration takes place *between* service invocations, not during the invocation processing. Given that the components are stateful, *session objects*, familiar from web programming, should be the data unit for migration [7]. Data elements are not migrateable if they represent operating system resources like open files, sockets or locks (cf. the `serializable` interface in Java).

Migration of service components requires the implementation of callback methods for life cycle management, since only the component itself will know how to prepare its session state into a representation fit for migration.

D. Resource Needs and Load Predictability

Since the grid of spacecrafts are orbiting the Earth in a predictable manner, both the available communication links and the expected *offered load* from surface clients can be estimated in advance. The population density and technological advancement of any area of the Earth is well known, so the expected offered load can be estimated based on position and time (night/day, etc.), or subject to a machine learning algorithm.

The population density on Earth is highly concentrated within small areas, and an orbiting spacecraft will spend most of its time over uninhabited areas. Figure 1 shows the population number within the footprint of a satellite during three consecutive orbits. It has been an essential idea in the SIN study that busy satellites should be able to share their workload with idle satellites in the vicinity [5].

Due to these properties, there is less need for discovery protocols related to link or peer availability. *Service discovery* mechanisms are likely still to be necessary since the migration and activation pattern of *services* are independent from orbital elements and population density data.

E. Fail-over Arrangements

What is not predictable, however, are fail and crash of services or entire spacecrafts. A simplistic fail-over arrangement would be to redirect client requests to redundant servers, while a more elaborate approach would also deal with the recovery of atomic transactions through, e.g., checkpoints or idempotent operations. For many applications, the simplistic approach suffices. The fail detection mechanism must produce the same result for all clients, for the sake of sharing and cooperation between clients. The fail management should therefore be conducted in the system/network management plane and the necessary fail-over information distributed to all spacecrafts.

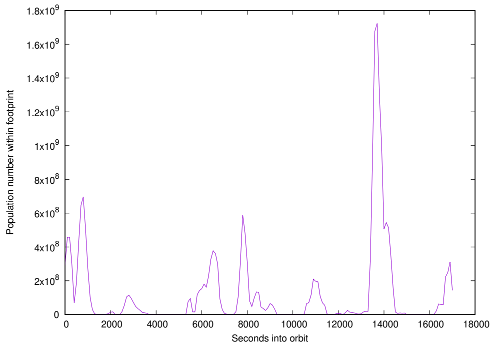

Figure 1. The population number inside the footprint of a satellite during three subsequent orbits.

F. Security and Trust Management

The SIN exposes a high number of service access points, and surface clients as well as customer code running in services inside the spacecrafts are not to be trusted. The links between spacecrafts running SIN-OS instances carry application traffic, as well as communication related to management and maintenance (cache replication, state migration, software updates, etc.). Application and administrative communication must be kept strictly separate through, e.g., Virtual Private Network (VPN) technology. VPN also contributes to relaxed IP address management for SIN customers.

With regard to trust management (a term that includes credential management and validation), the analysis published in 2021 [4] concluded that (1) the standard PKI model is not well suited due to connectivity and capacity demands, and (2) that both authentication and authorization control should happen in the same invocation, using the same set of credentials [9].

III. SIN-OS OVERVIEW

The different software components of a SIN-OS are shown in Figure 2 with their relations indicated by arrows. The executing component both on the server and the client side has been placed in *containers*, as a middleware layer for useful abstractions of the host API, as well as the control of the component's life cycle. On the client side, the management of handover operations is likely to demand a cross-layer connection to the radio hardware in order to detect when a handover is necessary. Event notifications are found in the Component API, for life cycle management purposes. No need for event notifications from the host OS to the container was identified at this stage of study.

The Connection Management and Communication Subsystem, shown in the bottom part of Figure 2, are comprehensive components which handle packet forwarding, route planning, handover planning and execution, fail-over execution, etc. For the services offered by the SIN-OS, shown on the right side of Figure 2, the following comments apply:

Non-Volatile Storage
: For storage of data across the duration of client sessions. This service is more than a simple file service, since it may scatter data on several storage tiers based on their access frequency. The API for the service may employ different access semantics: Flat files, Relational Database Management System, Tuplespace, etc.

Shared data segments
: Clients should be able to collaborate through shared data segments offered through a service interface, which ensures the chosen semantic properties: transactional context, update ordering, etc. The service may choose to spread the data across several spacecrafts according to their access frequencies in order to reduce the overhead of handover operations [8].

Cooperative caching
: A caching service used by one or more clients for lookup on immutable data elements. The service employs a cluster of neighborhood satellites for load balancing purposes, and replication of data between cache clusters to improve cache hit-rate [5].

Session State objects
: The application component may keep its session state in a separate *session object*, which must be migrated to new nodes subject to a handover operation. It is the responsibility of this service to associate a running session with a specific object to determine the migration operation. Studies have shown that, in the same manner as a shared object, the data elements can be "left behind" during handover and migrated on demand as they are being accessed from the new space location [7].

Discovery Services
: Application components may need to invoke services elsewhere in the SIN. The interface of the dependent service is known at compile time and necessary stubs generated, etc., but the location of their endpoint must be determined in run time. The discovery service serves this purpose, and application services notify the discovery service about their new endpoint as they are migrated to new endpoints.

Certificate & Key store
: Certificates should be kept in a safe storage after they have been validated, and private keys should not be exposed outside a trusted environment. This particular service is shown not to reference the communication subsystem, it is a local service which offers a client to sign a hash value or a secret key with the encapsulated private key. The implementation of this service has not been decided, but should employ suitable hardware based solutions (e.g., the Trusted Platform Module).

IV. API COLLECTION

The different software components involved in the SIN service will need APIs related to the specific tasks that the component is assigned to. Three distinct APIs will be briefly presented, together with a suggested set of service calls. Please observe that there is no call to establish an authenticated session, so necessary credentials and validation parameters for two-ways authentication and authorization control must be given as parameters in the service call. The reason for this design choice is to keep transactions atomic and idempotent. If either of the two parties fail to provide necessary credentials the actual service call will not be executed.

A. Client API

The client API is implemented as a container layer in the surface based client computers, it serves requests from "end" clients (clients that do not provide services to satellite based processes). The same API is used by application *owner*, who is allowed to start/stop/update the service, and application *user*, who are allowed to connect/invoke the service. The proposed service calls are:

`uploadApplication`
: Deploy new and updated applications

`startApp, stopApp`
: Reserved for the application owner

`connectApp, invokeService`
: Used by application user

`requestHandover`
: A handover is not initiated by user commands, but by the communication stack

B. Container API

The container is responsible for the creation of a runtime environment for the service application component, as well as the interface to the host resource management and the migration of components. The API offered to the container by the SIN-OS will not include calls across the interface between the container and the component. These calls will be introduced with the component API. The suggested calls are:

`loadApp, startApp, suspendApp, destroyApp`
: Call to allocate resources and load code segments

`executeHandover`
: Call to the host to find a new candidate service and to move the state representation there. The container must identify to the host the resources that must be migrated.

The container architecture is inspired by the Docker Swarm project [10], but its simplistic approach to load balancing, where the requests are distributed without regards to the networking costs/latency, must be replaced with a mechanism which takes the workload on intermediate nodes into consideration.

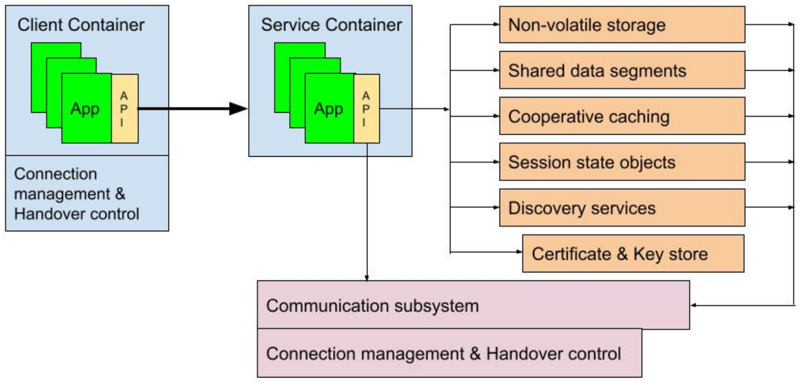

Figure 2. The components of a SIN-OS and their relations

C. Component API

The application components need access to services essential to their execution, including

`open, read, write, append, close, delete`
    Access to non-volatile memory
`open, close, read, write`
    Access to shared segments [8]
`open, lookup, add, close, delete`
    Creation/access to cooperative caches [5]
`findService, invokeService`
    Discovery/invocation of dependent services
`socket, read, write, close`
    Access to communication sockets
`init, destroy, suspend, resume`
    Life-cycle management callbacks

For service components, compile-time resources are also needed for the access to dependent services: Naming convention, stub object generations, etc.

V. The Software Model

This position paper presents a design proposition for a SIN-OS design, but the design should be subject to a closer study through a realistic software model of the satellite constellation. Previous efforts in this series of SIN studies have employed a software model programmed in Java for this purpose. A screenshot of this model is shown in Figure 3 for a constellation of 150 satellites at 500 km altitude. The colored backdrop in the figure indicates the population density inside the satellite footprint at a given location, based on gridded population data from NASA [11]. This data set has also been used to calculate the graph in Figure 1. The author prepares this model with additional logic for testing the API design as a further study.

VI. Conclusion

This article has proposed a design for a SIN-OS, based on a series of studies into a range of operational problems related to SIN-OS operation. Both a component/service map and a list of APIs have been presented. The proposed design is a part of an ongoing feasibility study on SIN development, and

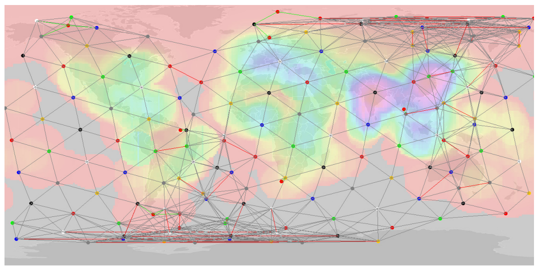

Figure 3. Screenshot from the satellite constellation model.

there are still many details in need for a further study. This will be the focus for further research effort in the field of SIN operation.

References

[1] S. Briatore, N. Garzaniti, and A. Golkar, “Towards the internet for space: Bringing cloud computing to space systems,” in *36th International Communications Satellite Systems Conference (ICSSC 2018)*, 2018, pp. 1–5.

[2] L. Bai, T. de Cola, Q. Yu, and W. Zhang, “Space information networks,” *IEEE Wireless Communications*, vol. 26, no. 2, pp. 8–9, 2019.

[3] A. Fongen, “Application services in space information networks,” in *CYBER 2021*. Barcelona, Spain: IARIA, Oct 2021, pp. 113–117.

[4] A. Fongen, “Trust management in space information networks,” in *SECURWARE 2021*. Athens, Greece: IARIA, Nov 2021, pp. 14–18.

[5] A. Fongen, “Cooperative caching in space information networks,” in *INTERNET 2022*. Vienna, Italy: IARIA, May 2022, pp. 1–5.

[6] A. Fongen, “Population-based routing in leo satellite networks,” in *MOBILITY 2022*. Porto, Portugal: IARIA, June 2022, pp. 1–4.

[7] A. Fongen, “Transfer of session state between satellites in a space information network,” in *INTERNET 2023*. Barcelona, Spain: IARIA, March 2023, pp. 1–4.

[8] A. Fongen, “Data sharing services in a space information network,” in *EMERGING 2023, The Fifteenth International Conference on Emerging Networks and Systems Intelligence*. Porto, Portugal: IARIA, September 2023, pp. 1–4.

[9] A. Fongen, “Optimization of a public key infrastructure,” in *IEEE MILCOM*, Baltimore, MD, USA, Nov 2011, pp. 1440–1447.

[10] A. Modak, S. D. Chaudhary, P. S. Paygude, and S. R. Ldate, “Techniques to secure data on cloud: Docker swarm or kubernetes?” in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 7–12.

[11] “Gridded population of the world v.4.11,” [Online; retrieved 8-Oct-2023]. [Online]. Available: https://sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse