



Carrier Grade Operating Systems

Gap Analysis

Version 2.0, October 23, 2007

Copyright © 2007 SCOPE Alliance. All rights reserved.

The material contained herein is not a license, either expressed or implied, to any IPR owned or controlled by any of the authors or developers of this material or the SCOPE Alliance. The material contained herein is provided on an “AS IS” basis and to the maximum extent permitted by applicable law, this material is provided AS IS AND WITH ALL FAULTS, and the authors and developers of this material and SCOPE Alliance and its members hereby disclaim all warranties and conditions, either expressed, implied or statutory, including, but not limited to, any (if any) implied warranties that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

Also, there is no warranty or condition of title, quiet enjoyment, quiet possession, correspondence to description or non-infringement with regard to this material. In no event will any author or developer of this material or SCOPE Alliance be liable to any other party for the cost of procuring substitute goods or services, lost profits, loss of use, loss of data, or any incidental, consequential, direct, indirect, or special damages whether under contract, tort, warranty, or otherwise, arising in any way out of this or any other agreement relating to this material, whether or not such party had advance notice of the possibility of such damages.

Questions pertaining to this document, or the terms or conditions of its provision, should be addressed to:

SCOPE Alliance,
c/o IEEE-ISTO
445 Hoes Lane
Piscataway, NJ 08854
Attn: Board Chairman

Or

For questions or feedback, use the web-based forms found under the Contacts tab on www.scope-alliance.org

Table of Contents

1 Purpose	3
2 Audiences	4
3 References	4
4 Overview	5
5 Terms and Definitions	6
6 Gap Summary	10
7 Gap Descriptions	13
7.1 Availability	13
7.1.1 Fault-Resistant File System (CGOS-1.1)	13
7.2 Performance	16
7.2.1 OS Tunable Parameters Enhancements (CGOS-2.1)	16
7.3 Security	16
7.3.1 Trust Mechanisms (CGOS-3.1)	16
7.3.2 Signed Executables (CGOS-3.2)	17
7.3.3 Unified Cryptographic Framework (CGOS-3.3)	18
7.3.4 Role-Based Access Control (CGOS-3.4)	19
7.4 Serviceability	20
7.4.1 Efficient Process CPU Usage (CGOS-4.1)	20
7.4.2 Functional Conformance Validation with CGL 4.0 (CGOS-4.2)	21
7.4.3 Persistent Shared Memory (CGOS-4.3)	22
7.4.4 Coherent User and Kernel Tracing Framework (CGOS-4.4)	23
7.4.5 Coarse Resource Enforcement (CGOS-4.5)	24
7.5 Standards	24
7.5.1 IP Routing and Forwarding (CGOS-5.1)	24
7.5.2 IPv6 Extensions (CGOS-5.2)	25
7.5.3 Layer 2 Tunneling Protocol Support (CGOS-5.3)	26
7.6 Hardware	26
7.6.1 Discovery of Platform CPU Architecture (CGOS-6.1)	26
7.6.2 Latency API for SMP / Multi-Core Programming (CGOS-6.2)	27
8 Conclusion	28
9 Appendix	30
9.1 Functional Considerations	30
9.1.1 Driver Hardening	30
9.2 Non-Functional Considerations	31
9.2.1 Application Binary Compatibility	31
9.2.2 Application Compatibility between Distributions	32
9.2.3 Extended Support Model	32
9.3 Erratum	33
9.3.1 POSIX Memory Protection	33

1 Purpose

A group of Network Equipment Providers (NEPs) formed the SCOPE Alliance with the intent of developing profiles for, and identifying gaps in, existing open specifications. The SCOPE Alliance further aims to prioritize the importance of implementing these various aspects of the specifications in the Carrier Grade Base Platform (CGBP) ecosystem. NEPs have an overriding requirement for hardware and software in their telecommunications applications and services, specifically, that they are “Carrier Grade” with all of the high availability, reliability, failover capability, serviceability, scalability and performance that this term implies.

More and more parts of Carrier Grade Base Platforms are specified by industry initiatives, and are built with existing hardware and software components. This trend enables the development of a vibrant ecosystem of suppliers of Commercial-Off-The-Shelf (COTS) hardware and software and Free Open Source Software (FOSS) from which NEPs can obtain a majority of their CGBP hardware and software components. Developing and manufacturing their own components can be a significant investment, which distracts NEPs from their focus of providing their customers with best-in-breed solutions.

To enable and encourage such a vibrant ecosystem, the SCOPE Alliance is identifying, prioritizing and publishing lists of suggested open standards, specifications and associated content that best enable the member companies of the SCOPE Alliance to deliver solutions that fit their customers’ needs. The published profiles identify the key capabilities required from each open standard or specification. Well-defined profiles aim to encourage the broadest possible ecosystem of suppliers from which to choose CGBP hardware and software components — characterized by multiple vendors, interchangeability and compatibility of components, and application portability. The SCOPE Alliance is also identifying gaps in existing open standards and specifications.

This document discusses enhancements required for operating system products to be appropriate for the CGBP environment. Many operating system vendors, irrespective of whether they have a Carrier Grade Linux (CGL) offering, have used the CGL specifications as a surrogate for the NEPs’ requirements. The CGL specifications provide lists of attributes that a Linux distribution is expected to implement.

This document identifies gaps in the Linux Foundation (formerly OSDL) CGL 4.0 specification. The term gap, as used in this document, is a feature that is not included in the CGL 4.0 specification but that the SCOPE Alliance strongly believes should be added to the CGL specification. If such a feature is included in open source projects outside the stock Linux kernel [31], it is still regarded as a gap.

The intent of this document is three-fold:

- To highlight gaps in the Linux Foundation CGL 4.0 specification.
- To provide guidelines and direction to the Linux Foundation, CGOS vendors and other industry or standards bodies.
- To foster the creation of projects by:
 - Carrier Grade Operating System vendors
 - Network Equipment Providers

- The Linux Foundation.

This CGOS Gap Analysis document is a living, changeable document that will be the basis for further investigations of the Linux operating system by the SCOPE Alliance.

2 Audiences

This CGOS Gap Analysis document is intended for the following audiences:

- Board and module vendors that use a Carrier Grade Operating System in their Network Elements and NEP applications built on Carrier Grade Base Platforms
- Carrier Grade Operating System implementers and providers
- The Linux Foundation, other specification bodies, special interest groups and related trade associations, which might find this information useful for defining new requirements or developing modifications to existing requirements
- The open source community at large.

3 References

1. J. N. Herder, H. Bos, B. Gras, P. Homburg and A. S. Tanenbaum, Failure Resilience for Device Drivers, Proceedings of the 37th IEEE/IFIP International Conference on Dependable Systems and Networks, Edinburgh, UK, June 2007.
2. IETF RFC 2307, LDAP, <http://www.ietf.org/rfc/rfc2307.txt>
3. IETF RFC 2401, IPSec, <http://www.ietf.org/rfc/rfc2401.txt>
4. IETF RFC 2460, IPv6, <ftp://ftp.isi.edu/in-notes/rfc2460.txt>
5. IETF RFC 2661, L2TP, <http://www.ietf.org/rfc/rfc2661.txt>
6. IETF RFC 3530, NFSv4, <http://www.ietf.org/rfc/rfc3530.txt>
7. IETF RFC 3931, L2TPv3, <http://www.ietf.org/rfc/rfc3931.txt>
8. Linux Standard Base (LSB), <http://www.linuxbase.org>
9. Open Group Base Specifications, Issue 6, IEEE Std 1003.1, 2004 Edition, <http://www.opengroup.org/onlinepubs/009695399/helf/codes.html>
10. OSDL CGL Requirements Definition Overview, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-overview.pdf>
11. OSDL CGL Availability Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-availability.pdf>
12. OSDL CGL Clusters Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-cluster.pdf>
13. OSDL CGL Serviceability Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-serviceability.pdf>
14. OSDL CGL Performance Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-performance.pdf>
15. OSDL CGL Standards Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-standard.pdf>
16. OSDL CGL Hardware Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-hardware.pdf>
17. OSDL CGL Security Requirements Definition, Version 4.0, <http://developer.osdl.org/dev/cgl/cgl40/cgl40-security.pdf>
18. PICMG 3.0 – AdvancedTCA™ Base Specification, PICMG 3.0, Revision 1.0, December 2002, PCI Industrial Manufacturers Group


19. PKCS #11 Cryptographic Token Interface Standard, Version 2.20, June 2004, RSA Laboratories, <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf>
20. SCOPE AdvancedTCA™ Hardware Profile, Version 2.0, <http://www.scope-alliance.org/pr/SCOPE-ATCA-Profile-v2.0.pdf>
21. SCOPE AMC Port Map Gap Analysis, Version 1, <http://www.scope-alliance.org/pr/SCOPE-AMC-Port-Map-Gap-Analysis-v1.pdf>
22. SCOPE Carrier Grade Middleware Profile, Version 1.0, http://www.scope-alliance.org/pr/SCOPE_CG_Middleware_profile_v1.0.pdf
23. SCOPE Definition Profile, http://www.scope-alliance.org/docs/Services-Profile_Service-Availability_v1.0.pdf
24. SCOPE Linux Profile, Version 1.0, <http://www.scope-alliance.org/pr/SCOPE-linux-profile-v1.pdf>
25. SCOPE Linux Profile, Version 1.1, http://www.scope-alliance.org/pr/SCOPE_Linux_profile-v1.1.pdf
26. SCOPE Linux Profile, Version 1.2, <http://www.scope-alliance.org/pr/SCOPE-Linux-Profile-v1.2.pdf>
27. Scoping the SCOPE: Closing the Gaps of an Open Carrier Grade Base Platform, Version 1.1, <http://www.scope-alliance.org/pr/scope-technical-position.pdf>
28. Service Availability Forum, Application Interface Specification, http://www.saforum.org/specification/AIS_Information/
29. Service Availability Forum, Hardware Platform Interface, http://www.saforum.org/specification/HPI_specification/
30. Service Availability Forum, Systems Management Interfaces, <http://www.saforum.org/specification/SMS/>
31. Stock Linux Kernel, <http://www.kernel.org/>
32. The Linux Foundation, <http://www.linux-foundation.org>
33. The Linux Foundation, About Carrier Grade Linux (CGL) http://www.linux-foundation.org/en/Carrier_Grade_Linux

4 Overview

The Carrier Grade Operating System (CGOS) Working Group of the SCOPE Alliance has reviewed the Carrier Grade Linux (CGL) 4.0 requirements specification. The CGL requirements specification was written by the CGL Working Group that was originally part of the Open Source Development Laboratory (OSDL). This group is now a Linux Foundation Working Group. See <http://www.linux-foundation.org>

The CGL requirements specification intends to demonstrate the applicability of Linux to Carrier Grade (CG) environments. It intends to enhance that applicability by describing additional functionality required by such environments and to guide the community in producing future versions of Linux. The CGL requirements specification addresses the topics of availability, performance, security, serviceability, standards, hardware and clusters. The most recent version of the CGL requirements is CGL v4.0 [10]-[17].

Telecommunication systems and applications have critical requirements for reliability and quality. In the past, Network Equipment Providers (NEPs) achieved these requirements by producing their own CG environments that included custom operating systems and infrastructure. Today, every NEP is attempting to reduce costs by using COTS and FOSS operating systems and, to the maximum extent possible, infrastructure software. The SCOPE Alliance is attempting to highlight areas in COTS and FOSS operating systems that need to be improved to achieve the CG requirements. In many cases, these changes have general applicability, and will help the operating systems adopting them to become more resilient, scalable and capable.

	<p align="center">Carrier Grade Operating Systems</p> <p align="center">Gap Analysis v2.0, October 23, 2007</p>
---	---

Given that most CGOS vendors use the CGL requirements specifications to identify capabilities to include in their offerings, the SCOPE Alliance believes that identifying areas in those specifications that require more clarity or additional capabilities is the most useful mechanism for communicating the needs of the NEPs to those vendors.

5 Terms and Definitions

	Term	Definition
3DES	Triple Data Encryption	A block cipher formed from the Data Encryption Standard (DES) cipher by using it three times.
AES	Advanced Encryption Standard	A block cipher used for encryption; a successor to the Data Encryption Standard (DES).
API	Application Programming Interface	An interface that a computer system or program library provides to support requests for services from it by an application.
ATCA	Advanced Telecommunication Computing Architecture or AdvancedTCA™	A specification targeted at requirements for carrier grade communications equipment.
CDR	Call Detail Record	A record containing information relating to a single call or session.
CG	Carrier Grade	A term for public switched telecommunications network products and services that must provide superior reliability and quality.
CGBP	Carrier Grade Base Platform	A computer system comprising hardware and software (operating system and middleware) that satisfies the CG requirements.
CGL	Carrier Grade Linux	A set of requirements published by the Linux Foundation for availability, clusters, performance, security, serviceability, standards and hardware, in order for Linux to be considered ready for use within the telecommunications industry.
CGOS	Carrier Grade Operating System	An operating system that implements the CG requirements.
COTS	Commercial-Off-The-Shelf	Software or hardware products, that are ready-made and available for general sale and use.
CPU	Central Processing Unit	A component of a computer that is capable of executing a program, i.e., a processor.
DBE	Double Bit Error	Two incorrect bits in a word or a message.
DH	Diffie Hellman	A cryptographic protocol for key exchange that allows two parties to establish a shared secret key over an insecure communication channel.

DSA	Digital Signature Algorithm	A type of asymmetric cryptographic algorithm used to provide authentication in digital (rather than written) form that uses a private key for signing a message and a public key for verifying the signature.
ECC	Error Correcting Code	A method for correcting errors that is used in data storage and data transmission, often without the operating system being aware of it.
FOSS	Free Open Source Software	Software provided under an open source software license.
IKE	Internet Key Exchange	The protocol used to set up a Security Association (SA) in the IPSec protocol suite.
IP	Internet Protocol	A protocol in the Internet protocol stack that uses packet switching and that provides unique global addressing.
IPSec	Internet Protocol Security	A suite of protocols for securing IP communication by authenticating and/or encrypting IP packets.
JCE	Java Cryptography Extension	Java APIs for several encryption mechanisms.
L2TP	Layer 2 Tunneling Protocol	A protocol for tunneling network traffic that is used to carry PPP traffic and to support VPNs.
LDAP	Lightweight Directory Access Protocol	A protocol for querying and modifying directory services running over TCP/IP.
LF	Linux Foundation	A non-profit organization dedicated to the advancement of the Linux kernel; formed in January 2007 as a merger of the OSDL and the Free Standards Group. See http://www.linux-foundation.org
LSB	Linux Standard Base	Specifications that standardize the internal structure of Linux operating systems.
MC2	MF SHM MPR	Memory Mapped Files or Shared Memory Objects or Memory Protection (as defined below).
MF	Memory Mapped File	File mapping is the association of a file's contents with a portion of the address space of a process.
MPR	Memory Protection	A way in which an operating system controls memory usage on a computer to prevent processes from accessing the memory of other processes.
NEP	Network Equipment Provider	A company that provides telecommunications equipment.
NFS	Network File System	A computer's file system that supports sharing of files, printers and other resources as persistent storage over a computer network.
NIS	Network Information Service	A client-server directory service protocol for distrib-

		uting system configuration data such as user and host names between networked computers.
NUMA	Non-Uniform Memory Architecture	A computer memory design for multi-processors, where the memory access time depends on the memory location relative to a processor.
OS	Operating System	A set of computer programs that manage the hardware and software resources of a computer.
OSDL	Open Source Development Labs	See Linux Foundation (LF).
PICMG	PCI Industrial Computer Manufacturers Group	An organization of industrial computer manufacturers that produced the AdvancedTCA™ specification.
PPP	Point to Point Protocol	A protocol used to establish a direct connection between two nodes.
PSM	Persistent Shared Memory	An area of memory containing information that is available across operating system reboots.
RADV	Router Advertisement	A list of a router's addresses on a given interface and their preference for use as a default router.
RAID	Redundant Array of Independent Disks	An umbrella term for data storage schemes that divide and/or replicate data among multiple hard drives.
RBAC	Role-Based Access Control	An approach to restricting system access to authorized users.
RSA	Rivest Shamir Adleman	An algorithm for public key cryptography that can be used for both encryption and digital signatures, which involves a public key for encrypting messages and a private key for decrypting them.
SAD	Security Association Database	A database that contains Security Associations (SAs), where a SA is a set of security information that describes a particular kind of secure connection between two devices.
SBE	Single Bit Error	One incorrect bit in a word or a message.
SHA1	Secure Hash Algorithm 1	A cryptographic hash function that computes a fixed-length digital representation (digest) of a message.
SHM	Shared Memory Object	An object that represents memory that can be mapped concurrently into the address spaces of multiple processes.
SMP	Symmetric Multi-Processor	A multi-processor computer architecture in which two or more identical processors are connected to a single shared main memory.

SMT	Simultaneous Multi-Threading	A technique for improving the efficiency of super-scalar CPUs that permits multiple independent threads of execution to better utilize the resources.
SNMP	Simple Network Management Protocol	A protocol used by network management systems to monitor network-attached devices for conditions that warrant administrative attention.
SPD	Security Policy Database	A database that contains Security Policy (SPs), where a SP is a rule that is programmed into the IPSec implementation that tells it how to process datagrams that a device receives.
SSH	Secure Shell	A protocol that allows data to be exchanged over a secure channel between two computers, using encryption and authentication.
SSL	Secure Sockets Layer	Cryptographic protocols that provide secure communications over the Internet; the predecessor of Transport Layer Security (TLS).
TPM	Trusted Platform Module	A facility for secure generation and use of cryptographic keys.
VLAN	Virtual Local Area Network	A method of creating independent logical networks within a physical network.
VPN	Virtual Private Network	A communication network tunneled through another network, and dedicated to a specific purpose.
VRF	Virtual Routing and Forwarding	A technology used in computer networks that allows multiple instances of a routing table to co-exist within the same router at the same time.

6 Gap Summary

The SCOPE Alliance CGOS Working Group has identified 16 gaps in the CGL specifications. Some of these capabilities exist in certain operating system releases but not in all of them. In other cases, they appear to be missing in almost all commonly used operating system products.

The gaps presented in this document identify features that the SCOPE Alliance regards as requirements that should be added to the CGL specifications. In the terminology of the CGL specification, all of the gaps are regarded as “mandatory” throughout this document, the gaps are prioritized as High, Medium and Low, which are defined as follows:

- High - Start implementation now.
- Medium - Start implementation as soon as possible.
- Low - Start implementation as soon as the gaps prioritized as High and Medium have been implemented.

The priorities of the gaps give advice to the community on their relative importance and the order in which to implement them, as the Scope Alliance sees it.

The gaps are shown below in tabular form grouped according to the existing CGL specification taxonomy, which comprises the following seven categories:

1. Availability
2. Performance
3. Security
4. Serviceability
5. Standards
6. Hardware
7. Clusters.

The categories into which these gaps are placed are recommended, rather than mandatory. There are no gaps for the Clusters category.

Detailed descriptions of the gaps are presented in Section 7 of this document.

Availability			
GAP ID	Name	Description	Priority
CGOS-1.1	Fault-Resistant File System	To provide a robust fault-resistant file system that can maintain file and data integrity, despite faults and reboots that occur while the data are being upgraded.	Medium

Performance			
GAP ID	Name	Description	Priority
CGOS-2.1	OS Tunable Parameters Enhancements	Detailed documentation for operating system tunable parameters, and notification when thresholds are exceeded.	High

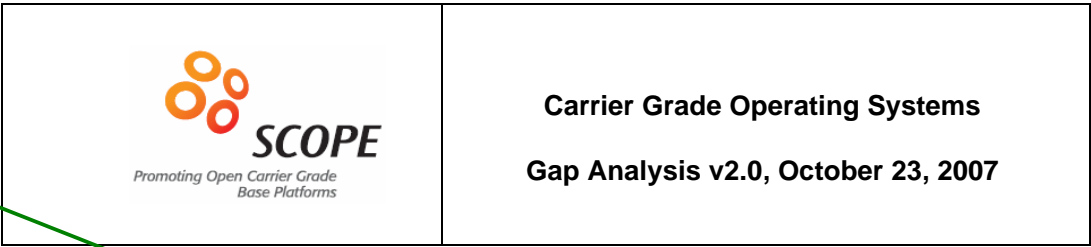
Security			
GAP ID	Name	Description	Priority
CGOS-3.1	Trust Mechanisms	Support for basic trust mechanisms, including secure boot, remote attestation, secure storage.	Medium
CGOS-3.2	Signed Executables	Validation of software images before use.	Medium
CGOS-3.3	Unified Cryptographic Framework	Framework that supports encryption and message hashing for both kernel and user applications, secure tamper-proof storage for security-relevant data, and registration of cryptographic capabilities.	Medium
CGOS-3.4	Role-Based Access Control	Support for the notion of a role with a name and a set of commands, along with the abilities to assign a set of privileges when commands are executed, to assign a list of users authorized to assume a role, and to log and audit role actions.	Medium

Serviceability			
GAP ID	Name	Description	Priority
CGOS-4.1	Efficient Process CPU Usage	To provide a summary of overall CPU usage for highly threaded applications, including user, system and interrupt mode execution.	Medium
CGOS-4.2	Functional Conformance Validation with CGL 4.0	A set of tests that can verify functionality of the requirements of the CGL 4.0 specification.	High
CGOS-4.3	Persistent Shared Memory	Reserves a section of persistent shared memory for critical data, so they are available after system reboot, which is useful for diskless systems.	Medium
CGOS-4.4	Coherent User and Kernel Tracing Framework	Lightweight framework for use in production systems that incorporates a unified view of user and kernel tracing.	Medium
CGOS-4.5	Coarse Resource Enforcement	Resource usage enforcement on a larger than per-process basis such as a user ID or some other meta-object.	Low

Standards			
GAP ID	Name	Description	Priority
CGOS-5.1	IP Routing and Forwarding	Support for Virtual Routing and Forwarding for the Internet Protocol.	High
CGOS-5.2	IPv6 Extensions	Extensions to IPv6 including NFS and NIS.	Medium
CGOS-5.3	L2TP Support	Support for the Layer 2 Tunneling Protocol.	Medium

Hardware			
GAP ID	Name	Description	Priority
CGOS-6.1	Discovery of Platform CPU Architecture	Discovery of the topology and other details of a platform's CPU architecture, such as the number and the sizes of the caches, to facilitate SMP programming.	Low
CGOS-6.2	Latency APIs for SMP / Multi-Core Programming	Support for the notions of latency domain and locality domain, and APIs that allow a process to determine locality domain characteristics, such as the memory latency, and the communication latency between processes.	Medium

- Noted in open session that many of the required features will be present in next gen Linux filesystems (ext4? BTRFS?)



- Nortel suggested their TRFS might be a possible candidate as well but wanted to evaluate BTRFS to determine the suitability to their needs or the possible contribution of TRFS back to the community

- Also noted in the open session that filesystem contributions might be easier to get adopted into mainline than many other features

- Need a specific definition of "essential data" here. Data required to repair / rebuild FS? Some means of classifying data actually being written to the filesystem? Something else?

- NEW GAP

7 Gap Descriptions

7.1 Availability

7.1.1 Fault-Resistant File System (CGOS-1.1)

- CGL WG feels this requirement should be broken down into several individual requirements, probably one for each bullet point in the description.

7.1.1.1 Purpose

To provide a robust fault-resistant file system that can maintain file and data integrity, despite faults and reboots that occur while the data are being upgraded.

- What does "upgraded" mean here? Updated? Or is this a requirement for in-service FS driver upgrades?

7.1.1.2 Description

Applications and deployment scenarios in Carrier Grade telecommunication environments need to provide a robust fault-resistant file system for the CGOS, with:

- Data integrity protection by means of internal checksums
- Data integrity model that guarantees file system metadata and data consistency and fast recovery in the presence of incomplete updates due to unexpected reboots
- Data integrity model that minimizes the impact of corruption of essential data
- Online integrity / consistency checking and recovery facilities
- Protection from unexpected multiple accesses
- Resource allocation guarantees.

- NEW GAP: is this requirement met by locking at the filesystem / userspace level?

In addition, the file system for the CGOS needs to support continuous availability of service, so that the majority of administrative tasks can be performed online without service interruption, and also deployment and upgrade capabilities required by Carrier Grade telecommunication environments.

Data integrity models, consistency checks and recovery

The CGOS needs to provide support for a fault-resistant file system to ensure that the data presented to the applications are correct. Studies have indicated that disk drives have larger than expected failure rates. Even if a disk drive does not fail completely, marginal components within the system such as power supplies or components within the drive can cause intermittent failures, which generally are hard to reproduce.

Engineers writing file systems often assume that the underlying storage medium supporting the file system is reliable or that errors encountered in the storage medium are handled transparently by the medium. When these assumptions are not borne out in reality, problems can occur. Incorrect metadata can cause file system corruption or a system panic. Bad data presented to an application can lead to incorrect decisions in program logic, unexpected termination of programs, or programs that are not able to progress.

Mirroring data is not the complete answer, because mirroring requires that all copies of the data in the mirror are identical and correct, and that any mirror can be used with equal safety. A hardware Redundant Array of Independent Disks (RAID) can provide lim-

ited protection, but does not protect against errors introduced by the controller itself or on the path to / from the controller.

What is needed is end-to-end checksums of every in-use block. This requirement is achieved by computing and storing checksums for data, separate from the data itself, and by verifying the checksums when the data are read. In a mirrored environment, if a checksum fails to validate one of the mirrors, other mirrors can be consulted to find valid data. There are several approaches on how this error detection can be implemented. The file system itself can provide support for the checksums or the checksums can be implemented in a block layer solution as a logical volume manager or software RAID.

When the system reboots unexpectedly, it might leave system data partially updated. The file system needs to employ a data integrity model to guarantee that it can repair such in-progress updates for both file system data and metadata. If the file system incurs a power outage in the middle of a data write, committed data must not be lost and the file system must be recovered to a consistent state. There are several approaches possible and being employed in existing file systems. The file system can implement journaling or transactional copy-on-write operations, or can ensure data integrity using ordering of updates (e.g., soft updates). What is required is that data integrity protection is available for both file system metadata and data.

Recovery from partially completed updates (with whatever data integrity model) does not provide protection from corruption caused by disk errors, file system bugs, administrator errors, or other sources. In addition to file system data integrity protection capabilities outlined above, the CGOS needs to provide online consistency / integrity checking and recovery.

The impact of potential data corruption must be minimized with a data integrity model that employs replication of essential data (e.g., mirroring). For example, the loss of a single superblock due to disk hardware faults should not cause catastrophic consequences and render the entire file system data unavailable.

For availability models employed in typical telecommunication environments, there is a strong emphasis on shared storage. The CGOS file system must be able to protect itself from corruption caused by unintentional simultaneous access by multiple users in a shared disk environment.

Support for allocation reservations

Often there is a need for applications to pre-allocate space for file(s) in a file system. Applications can use this feature to avoid fragmentation to a certain extent and, thus, obtain faster access. With pre-allocation, applications also obtain a guarantee of space for particular file(s), even if the file system becomes full later.

POSIX defines the *posix_fallocate()* function, which can be used for a similar purpose. For traditional Linux implementations, this function is quite slow (because it writes zeroes to each block that must be pre-allocated). File systems can achieve this pre-allocation more efficiently within the kernel. It is expected that *posix_fallocate()*, or its equivalent, will be modified to benefit from this capability.

- NEW GAP: sounds like this is a concern about the performance of *posix_fallocate()* and the requirement is to have the functionality as part of the VFS layer?

- NEW GAP: doesn't seem to be covered above.

- Should be a separate gap.

- Reword title to remove "defragmentation" since the requirement is to reduce or eliminate FS fragmentation.

Online defragmentation

Most file systems are susceptible to performance degradation over a period of time due to fragmentation of on-disk data, although to different degrees, depending on the specific file system implementation. The seek performance of traditional mechanical hard drives is not increasing as rapidly as the disk capacity, i.e., when normalized to capacity, disks are getting slower. The performance of disks has become increasingly important for the current generation of disks, processors and networks. The CGOS needs to provide provisioning to reduce the effects of internal fragmentation, either by design or with the help of online defragmentation facilities or preferably both.

Support for file system snapshots / clones

- NEW GAP: Does BTRFS meet this requirement?

Snapshots are useful for a variety of backup / upgrade scenarios.

Snapshots provide a useful solution for a "backup window" problem. Full backups usually take considerable time and are not transactional and atomic. Creation of a snapshot is usually takes $O(1)$ time, while performing a direct backup usually takes $O(\text{size of data})$ time.

Writable snapshots (sometimes called "clones") can be useful in the implementation of a rollback mechanism in certain upgrade scenarios.

There are several approaches to implement snapshot functionality. Snapshot capabilities can be implemented as an integral part of the file system or can be provided by a block device layer (e.g., a logical volume manager or certain hardware RAIDs). Even in the latter case, some cooperation with the file system layer / tools is needed, because snapshots are expected to be taken while the file system is in a quiescent state, e.g., by committing pending transactions and holding new ones until the snapshot is taken.

Although the implementation of snapshot capabilities at the file system layer is not a prerequisite, it might have certain advantages because block-level snapshots are almost always less space-efficient than direct file system support for snapshots.

- NEW GAP: Filesystem needs to be able to grow without being unmounted? Does BTRFS do this already?

Online resizing

As the demand for capacity grows, there are number of scenarios in which the ability to expand / resize the file system online (e.g., without service interruption) becomes important.

Multi-architecture support

The file system metadata must be designed to be agnostic to the host CPU word length and endianness. As a result, the same file system image can be used by CPUs of different hardware architectures (e.g., ia32, x86-64, SPARC, etc), which is useful in certain hardware upgrade / migration scenarios.

7.1.1.3 Priority

Medium

- NEW GAP: Requirement appears to be "take hard disk A with filesystem CGOSFS partitions on it out of a machine with x86 processor, plug it in to a machine with SPARC processor, for example, and the data should still be read correctly". Is this correct?

- During open session it was suggested CGL document what tunable parameters are there today, provide that list back to SCOPE and ask for GAPS for all missing parameters.


- Discussion in working session suggested out-of-date or partial documentation was worse than none, so we should turn the whole requirement back to SCOPE with following suggestions:

1. Is there known parms that are not documented or documented wrong? Please list.

2. Please rewrite GAP avoiding "all supported" language as all supported tunables in kernel are documented in LKC.

3. Please provide more context as to what options must be documented and how. Is 'sysctl -a' adequate?

- Split into new gaps for each required feature.

 <p>Carrier Grade Operating Systems Gap Analysis v2.0, October 23, 2007</p>
--

7.1.1.4 CGL Specification

Availability

7.2 Performance

7.2.1 OS Tunable Parameters Enhancements (CGOS-2.1)

7.2.1.1 Purpose

To document operating system tunable parameters and provide interfaces for delivering notifications when thresholds associated with such parameters are exceeded.

7.2.1.2 Description

The CGOS needs to provide well-documented tunable parameters and interfaces to satisfy critical performance requirements.

Operating systems offer a set of variables whose values can be manipulated to affect system performance. The CGOS requires the following enhancements.

The complete set of variables whose manipulation is supported must be documented. This documentation includes the variable type, default value, range of values supported by the operating system, variable's purpose, and external performance indications that a change from the default value would be beneficial.

In addition to knowing how the variables can be tuned, it is important to know when they should be tuned. Interfaces and mechanisms are needed that provide notifications when thresholds associated with the variables are exceeded. These interfaces and mechanisms must be subscription-based, with notifications provided only to those that subscribe for them.

The operating system tunable parameters should include all features used by the CGOS distribution.

7.2.1.3 Priority

High

7.2.1.4 CGL Specification

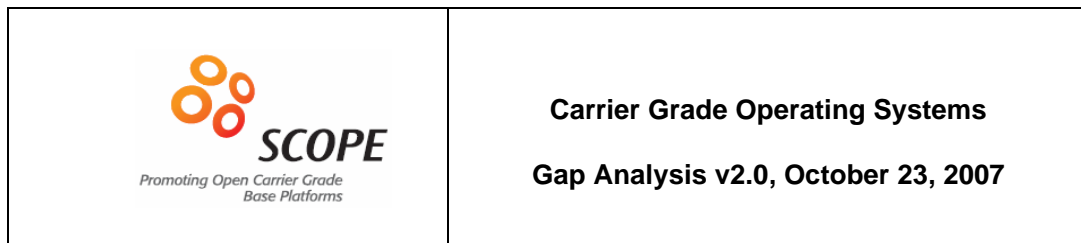
Performance

7.3 Security

7.3.1 Trust Mechanisms (CGOS-3.1)

7.3.1.1 Purpose

To support basic trust mechanisms, including secure boot, remote attestation and secure storage.



7.3.1.2 Description

The CGOS, and the underlying hardware platform, must provide basic trust mechanisms, so that a network element has assurance that no one has tampered with it. In particular, the CGOS needs to provide the following mechanisms.

- Secure boot: All software (including the BIOS and the OS) must be measured before being loaded or used. After measurement, the system can compare the measured values with pre-stored values and abort the loading if they do not match, or at least store the values in a trusted place for later evaluation.
- Remote attestation: In a networked environment, there is a need to ascertain the integrity of network peers. This feature can be used to exclude compromised base stations from the network.
- Secure storage: Both of the above mechanisms require secure storage, but secure storage also has other uses. It can be used to store communication keys for secure protocols, measurement values for both software and hardware, software that should not be reverse engineered, and critical logging data. The contents of the secure storage must be available only to trusted software that has been verified during the secure boot.

The Trusted Computing Group's Trusted Platform Module (TPM) is one way to implement these mechanisms, although there might also be other ways. If TPM hardware support is not available, these mechanisms can be implemented with other hardware solutions. Approximate solutions (i.e., not entirely secure solutions) can be implemented using software; in some cases, such solutions might be better than nothing.

The SCOPE Alliance recognizes that additional definition is required around the details of this gap; therefore, the Alliance welcomes the opportunity to collaborate with industry ecosystem partners and specification groups to develop additional specificity in this area.

7.3.1.3 Priority

Medium

7.3.1.4 CGL Specification

Security

7.3.2 Signed Executables (CGOS-3.2)

7.3.2.1 Purpose

To ensure that the software images being used are those that were intended to be used, without modification or corruption.


7.3.2.2 Description

The CGOS needs to provide the ability to ensure that the system is using the software distributed by a NEP. See also Trust Mechanisms (CGOS-3.1).

First, it is necessary to ensure that the operating system image being booted has not been modified since being deployed to where the equipment is located, or since being

- need specific use cases, rewrite to address specific attack vectors.
- signed binaries will not protect against environment variables or configuration file based attacks
- is it expected that this would be implemented at the fork/exec level of the kernel or somewhere else? What implementations are acceptable?

- Definitely a gap, but unclear what is required for it, need more context.
- Will TPM keys meet the GAP?
- Is this intended to address booting diskless nodes on untrusted networks?

	<p style="text-align: center;">Carrier Grade Operating Systems</p> <p style="text-align: center;">Gap Analysis v2.0, October 23, 2007</p>
---	---

reloaded from a NEP distribution. This checking must be incorporated within the boot process.

Additional checking must be performed to verify that all of the images deployed to a location have the contents distributed by the NEP. These contents include operating system images, executable images, file contents and libraries. A desired extension is the ability to verify an image each time it is loaded into memory for execution. It is also desirable that the image should be verifiable subsequently during execution.

The ability to sign executables is provided by computing a secure hash of the executable content, embedding the hash value into the executable, and later verifying that the signature reflects the original contents of the executable.

The signatures use public-key cryptography where the hash is computed with a private key held secret by the signer. The signer provides a public key as part of the software release, which can be used to verify, on request, the integrity of the executable.

In essence, utilities used to sign and verify executables must support the idea of a user producing signed executables and verifying those signatures with the appropriate private and public keys.

A future extension would optionally allow the operating system to verify the signatures of all (or some) of the signed executables before executing them.

Care must be taken to ensure that certificates that have been revoked are not used.

7.3.2.3 Priority

Medium

7.3.2.4 CGL Specification

Security

7.3.3 Unified Cryptographic Framework (CGOS-3.3)

7.3.3.1 Purpose

To provide a cryptographic framework that supports encryption and message hashing for both kernel and user applications, secure tamper-proof storage for security-relevant data such as keys, and registration of cryptographic capabilities.

7.3.3.2 Description

The CGOS needs to provide a unified framework for optimized implementations of common cryptographic (encryption and message hashing) algorithms.

Carrier grade solutions rely on communication protocols that have stringent security requirements. Typically, these protocols are based on standard security application providers such as SSL, SSH, IKE and JCE.

Data integrity is accomplished through mechanisms (message hashing) that check that data transmitted across the network or stored on/retrieved from disk without encryption

- Noted in open session that there are many ways to accomplish this but no good "glue" currently.

- Working session asked for more detail: userspace library? kernel only? kernel keyrings? Something else required?

are not modified. Data confidentiality is accomplished through mechanisms (encryption) that convert the data to a form not easily reversible, before being transmitted or stored.

The use of both encryption and message hashing for data that are transmitted or stored demands a cryptographic framework that is available to both the kernel and user applications and that transparently makes use of whatever hardware encryption capabilities are available.

A prerequisite to the security capabilities described above is the ability to store in a secure, tamper-proof way security-relevant data, such as keys used to verify the integrity of downloaded data. Keys can be loaded during system assembly, and additional keys can be provided using a secure mechanism after the system is started. Such a mechanism is almost always a combination of hardware, operating system and firmware. See also Trust Mechanisms (CGOS-3.1).

A unified cryptographic framework must expose to security providers a common interface to algorithms not only for various encryption algorithms (at the very minimum 3DES and AES) but also for message hashing (MD5, SHA1), message signing (RSA, DSA, DH) and random number generation. See the RSA cryptographic token interface standard PKCS #11 [19].

Hardware acceleration is also desirable for carrier grade components that use encryption. The cryptographic framework must offer mechanisms whereby device drivers can register the cryptographic hardware. A device with a cryptographic capability (key store, encryption algorithm) must be able to register the capability with the cryptographic framework. Registration includes, for example, the type of cryptographic capability, available algorithms, and number of contexts. When a driver initializes, it must register any cryptographic capabilities possessed by the device(s) it controls.

When a kernel thread or user process requests that a particular algorithm be used, the cryptographic framework must try to use the most efficient implementation based on the availability of resources in a transparent manner.

Algorithms must be easy to export / import, Cryptographic keys must be easily reduced to 56 bits, or cryptography must be easy to switch off.

7.3.3.3 Priority

Medium

7.3.3.4 CGL Specification

Security


7.3.4 Role-Based Access Control (CGOS-3.4)

7.3.4.1 Purpose

To support the notion of a role with a name and a set of commands, along with the abilities to assign a set of privileges when the commands are executed, to assign a list of users authorized to assume a role, and to log and audit role actions.

- CGL proposes SELinux as the implementation of this gap and requests SCOPE create new gaps for specific functionality missing from SELinux.

- What is the specific requirement for 56 bits here and is it still relevant?

	<p style="text-align: center;">Carrier Grade Operating Systems</p> <p style="text-align: center;">Gap Analysis v2.0, October 23, 2007</p>
---	---

7.3.4.2 Description

The CGOS needs to provide the ability to assign a name and a set of commands to a role, such that those commands and only those commands can be executed while the role is assumed.

It must provide the ability to assign a set of privileges (if privileges are defined), or a user id / group id, when the commands comprising the role are executed.

It must also provide the ability to assign a list of users authorized to assume a role, and to require that roles are assumed, i.e., a user that has already been authenticated to the system must “log in” to the role using whatever authentication mechanisms are required.

All role actions, including assuming and releasing a role together with the required credentials, must be capable of being logged and audited by standard system auditing.

7.3.4.3 Priority

Medium

7.3.4.4 CGL Specification

Security

7.4 Serviceability

7.4.1 Efficient Process CPU Usage (CGOS-4.1)

7.4.1.1 Purpose

To provide a summary of overall CPU usage for highly threaded applications, where the summary includes user, system and interrupt mode execution.

7.4.1.2 Description

The CGOS needs to provide a summary of overall CPU usage for highly threaded applications, where the summary includes user, system and interrupt mode execution.

For threaded applications, the overall summary of CPU usage can be derived on demand by scanning all executing threads. For highly threaded applications, doing so is time consuming. For example, for a threaded application with 100 threads, such scanning will cycle through 100 task descriptors and sum the CPU usage of each thread. In addition, such scanning can enforce CPU limits on timer tick interrupts. Because the operation is so time consuming for highly threaded applications, enforcing CPU limits can be done only during tick processing. This mechanism can therefore be easily evaded (intentionally or not).

High availability software often relies on measuring overall process CPU usage.

To resolve this issue, it must be possible to measure the overall CPU time of a highly threaded application on the fly without any noticeable performance degradation. In addition, the measurement must satisfy the Precise Process Accounting requirement in the CGL 4.0 requirements specification. It must also facilitate fast retrieval of process time usage and enforcement of CPU exhaustion limits in context switching code.

- Action to the CGL WG to determine if this is already met by in-kernel information and if so, how it can be / already is exposed.

- Is this done via MSA?

- Is there some piece of PPA that needs to be implemented in MSA?

- Does CFS already collect all the required information?

Additional information, such as the time spent in user space waiting for locks and the time spent handling page faults, is also desired. These measurements cannot rely on periodic sampling. Each time a state transition occurs in a thread, the time must be recorded for each executing thread and the process containing the thread. Summing the usages of the individual threads might provide acceptable performance for a very small number of threads, but quickly becomes an unacceptable burden for a large number of threads.

7.4.1.3 Priority

Medium

7.4.1.4 CGL Specification

Serviceability

7.4.2 Functional Conformance Validation with CGL 4.0 (CGOS-4.2)

7.4.2.1 Purpose

To provide a set of tests that can verify the functionality of the requirements of the CGL 4.0 specification.

7.4.2.2 Description


There needs to be an available set of tests that can verify the functionality of the requirements of the CGL 4.0 specification. Some of these tests will be programmatic; some will be verification/checklist; and some will be manual.

NEPs need a mechanism to validate the contents of the CGOS when making design (and purchasing) decisions for network devices. CGOS distributors need a mechanism to validate the contents of their operating systems and to market or brand appropriately.

There is a need for a set of tests, scripts and best practices (collectively called the Validation Procedures) that are easy-to-execute and provide documented or reportable results. The Validation Procedures must be repeatable across multiple CGOS instances from different vendors, producing similar output or results. The Validation Procedures should be modular, so that requirements that are hardware-specific can be excluded to account for differences in the hardware. They should be as hardware-independent as possible and account for hardware when necessary. The Validation Procedures should take into account differences in requirements as follows:

- a) Some requirements are programmatically verifiable, but require complex tests. These tests must be created in a repeatable and scalable manner.
- b) Some requirements provide a broad overview of a feature. In these cases, a checklist is provided to focus on the functionality that must be certified. The checklist applies to all requirements, and describes the best practices to validate a feature. The report provides the supporting material that is missing in the requirements and that is needed for certification.

- Is this a request for the LSB Headless profile? Ask for clarification from SCOPE, present proposal for LSB Headless to see if that fulfills the gap.

	<p style="text-align: center;">Carrier Grade Operating Systems</p> <p style="text-align: center;">Gap Analysis v2.0, October 23, 2007</p>
---	---

- c) Some requirements need manual involvement and interpretation for validation, e.g., pulling hardware and checking the state of the system. An accompanying checklist documents the steps.
- d) Some requirements need vendor support to validate, e.g., SBE/DBE inducing hardware.
- e) Some requirements need support from modified firmware or boot loaders.
- f) Many requirements need manual certification by inspection of documentation.

The recommended approach to functional conformance validation is a phased approach, starting initially with performance and availability.

7.4.2.3 Priority

High

7.4.2.4 CGL Specification

Serviceability

7.4.3 Persistent Shared Memory (CGOS-4.3)

7.4.3.1 Purpose

To reserve a section of Persistent Shared Memory (PSM) that applications can use to house critical data, so that the data are available after the operating system reboots, which is particularly useful for diskless systems.

7.4.3.2 Description

The CGOS needs to reserve a section of Persistent Shared Memory (PSM) that applications can use to house critical data. The CGOS also needs to ensure that this PSM section is persistent across operating system reboots, i.e., the contents of the PSM are preserved and available to the applications after the operating system reboots. This feature is particularly useful for, and considered to be higher priority for, diskless systems.

Applications usually have some very critical data, such as call session records, billing records, logs, stack traces, pending messages, etc. Usually, this information is lost on a crash and reboot. Some parts of it might have been saved on persistent storage or transmitted to a management terminal, but other parts of it invariably get lost. Storing this information in a PSM section ensures that this information is available in memory even after the operating system comes up again. The information can be sent to a management terminal for further analysis after the operating system comes up again.

7.4.3.3 Priority


Medium

7.4.3.4 CGL Specification

Serviceability

- Need use-case clarification from SCOPE. Does warm reset case (ie. kexec) meet the requirement?

- if cold boot or full reset is required, will WR's pmem / pmemfs meet the requirement?

	<p style="text-align: center;">Carrier Grade Operating Systems</p> <p style="text-align: center;">Gap Analysis v2.0, October 23, 2007</p>
---	---

7.4.4 Coherent User and Kernel Tracing Framework (CGOS-4.4)

7.4.4.1 Purpose

To provide a low-overhead, flexible, integrated user and kernel tracing framework.

7.4.4.2 Description

The CGOS needs to provide a coherent user and kernel tracking framework that is safe to use in production systems.

The CGL 4.0 SFA.2.2 specification [13] discusses “Dynamic Probe Insertion” for the kernel. This model must be extended to include the ability to instrument user applications. This instrumentation may reside in only an application process or in both an application process and the kernel. For example, it should be possible to wait for an event to be captured by kernel tracing and then extract information from the process address space about what was happening when the event was triggered.

Tracing must be more capable than simply filling a ring buffer when a trace point is hit, copying the contents of the buffer, writing the data to a file, and then post processing it. The coherent user and kernel tracing framework must have the following characteristics:

- a) Tracing must not cause the system to panic. Panics must not be possible unless a very specific mode of operation is enabled. Even then, panics must be deliberate. There must be no unexpected side effects.
- b) It must be possible to insert static trace points into the kernel or user space that incur only the overhead of executing noops when they are not active. Thus, debug executables could be shipped as production code and trace points would be activated only when needed.
- c) Tracing must support the ring buffer method mentioned above, as well as the method whereby the kernel filters events of interest and delivers only those events that satisfy the filter criteria. Various options such as aggregation should be considered.
- d) It must be possible to have multiple trace sessions active simultaneously even at the same trace location.
- e) Tracing must be able to follow an activity across the user-kernel boundary.
- f) Tracing must be able to connect dynamically to the processes being traced. It must not be necessary to start a process under tracing.

7.4.4.3 Priority


Medium

7.4.4.4 CGL Specification

Serviceability

- Does kernel markers present in 2.6.25+ meet the requirement?

If not, send request back to SCOPE to define a specific list of dtrace-style use cases that are required. Ideally 10-20 total.

	<p style="text-align: center;">Carrier Grade Operating Systems</p> <p style="text-align: center;">Gap Analysis v2.0, October 23, 2007</p>
---	---

7.4.5 Coarse Resource Enforcement (CGOS-4.5)

7.4.5.1 Purpose

To provide the ability to impose resource consumption limits on one or more threads or processes.

7.4.5.2 Description

The CGOS needs to provide mechanisms that allow resource consumption constraints to be applied to an individual thread, a process and all processes running with a particular user ID or group ID, when resource consumption limits are exceeded.

These resource consumption constraints should follow today's mechanisms for resource exhaustion for individual processes and groups of processes.

Constraints must have actions that can be selected when an application is first started. Such actions include "log", "signal process" and "terminate process".

This requirement applies to CPUs as well as memory.

7.4.5.3 Priority

Low

7.4.5.4 CGL Specification

Serviceability

7.5 Standards

7.5.1 IP Routing and Forwarding (CGOS-5.1)

7.5.1.1 Purpose

To support Virtual Routing and Forwarding for the Internet Protocol.

7.5.1.2 Description

The CGOS needs to support Virtual Routing and Forwarding (VRF) for the Internet Protocol (IP), including both IPv4 and IPv6.

A key capability used by the NEPs is the ability to forward network packets using IP. This capability has lagged as new capabilities such as VPNs have emerged.

A network element that supports VRF contains multiple instances of routing and forwarding tables and looks like several independent routers to an external observer. At the routing protocol level, each VRF instance is a separate router with its own router identity and protocol state.

Each IP interface, e.g., Ethernet Virtual Local Area Network (VLAN), is associated with exactly one VRF instance and is invisible to the other instances. Each VRF instance has its own forwarding table, IPsec Security Association Database (SAD) and Security Policy

- per-thread ulimits

- still definitely a gap according to CGL

- noted in the open session that this seems like a reasonable request and could possibly be integrated into mainline if implemented.

- Needs clarification from SCOPE.

- Sounds like a gap to CGL.

- If this is VRF and VLAN, it is fully implemented today. Turn back to SCOPE to identify specific parts still missing.

- if it is not above and is instead networking namespaces this is a long way out and definitely a big gap.

Database (SPD), packet filtering rules, transport protocol termination, and local IP address space.

Each socket is associated with a VRF instance. A VRF-aware application can have open sockets to multiple VRF instances at the same time, for connecting to multiple networks, or for acting as an application layer gateway between networks. VRF does not require separation of applications into isolated environments corresponding to different VRF instances.

IP addresses used in different VRF instances can overlap because the instances are separate and independent.

7.5.1.3 Priority

High

7.5.1.4 CGL Specification

Standards

7.5.2 IPv6 Extensions (CGOS-5.2)

7.5.2.1 Purpose

To provide support for IPv6 in NFS and NIS.

7.5.2.2 Description

The CGOS needs to provide support for IPv6, so that infrastructure components exposed to the network can accept connections through IPv6, as well as IPv4. This capability includes support for IPv6 in the Network File System (NFS) and the Network Information Service (NIS).

IPv6 is the next generation of the Internet Protocol (IP). Movement to IPv6 is rapidly increasing. It must be possible to configure systems as only IPv4 systems, dual stack (IPv4 and IPv6) systems, or only IPv6 systems. Only IPv6 refers to both the ability to configure the system with only an IPv6 TCP/IP stack and the ability to enable all network facing applications that access TCP/IP to function in such an environment. Currently,

- NFSv4 (here v4 doesn't refer to IPv4) doesn't support IPv6. The source code doesn't have any IPv6 socket calls. See RFC 3530 [6].
- NIS doesn't support IPv6 but will probably be rendered obsolescent by the Lightweight Directory Access Protocol (LDAP) and/or Kerberos. The source code doesn't have any IPv6 socket calls. RFC 2307 [2] describes the alternative usage of LDAP.

The CGOS needs to provide support for IPv6 in NFS and NIS.

7.5.2.3 Priority

Medium

- Return to SCOPE.

- Break into two separate gaps: NFSv4 over IPv6 and NIS over IPv6.

- Both appear to still be gaps from CGL point of view, is NIS/IPv6 still a gap from CGOS PoV?

7.5.2.4 CGL Specification

Standards

7.5.3 Layer 2 Tunneling Protocol Support (CGOS-5.3)

7.5.3.1 Purpose

To support the Layer 2 Tunneling Protocol (L2TP).

7.5.3.2 Description

The CGOS needs to provide support for the Layer 2 Tunneling Protocol (L2TP).

Although TCP/IP over high-speed interconnects is at the core of today's telecommunications networks, other protocols, in particular L2TP, must be supported. L2TP is used to move Point to Point Protocol (PPP) packets across the network.

Thus, the CGOS needs to provide support for L2TP, i.e, it must support:

- RFC 2661 [5]

Moreover, it must support L2TPv3, but with lower priority, i.e., it must support:

- RFC 3931 [7].

7.5.3.3 Priority

Medium

7.5.3.4 CGL Specification

Standards

7.6 Hardware

7.6.1 Discovery of Platform CPU Architecture (CGOS-6.1)

7.6.1.1 Purpose

To allow the discovery of the topology and other details of a platform CPU architecture, such as the number and the sizes of the caches, to facilitate and optimize SMP programming.

7.6.1.2 Description

The CGOS needs to allow an application to discover platform CPU architecture topology and details, such as the number of caches and the sizes of the caches, to facilitate the optimization of the use of multiple CPUs, the memory hierarchy and the interconnect fabric. The CGOS needs to provide such architectural information in a format that is uniform across platforms.

Many forms of SMP are available today in CG environments ranging from SMT to NUMA and combinations in-between. The CPU configurations have a profound effect on performance and stability. The scheduler in most operating systems (Linux 2.6, for example)

- CGL has determined that L2TP is already fully implemented. Return to SCOPE, ask if the requirement is for PPTP? If so, it will remain a gap due to intellectual property issues relating to Microsoft's PPTP and the current, out-of-tree implementation.

- If both are required, this gap needs to be split into two separate gaps: L2TP and PPTP.

- CGL WG to document sysfs interface to determine this, call the gap complete and begin "implemented" phase of CGL/CGOS flowchart.

dynamically builds a view of the system based on the CPU topology, including caches and threads. This view must be exported to application programmers (to a certain degree some of that information is already available in /sys/devices/system/cpu directory).

The understanding of shared vs. private caches and threads is key to writing high performance software. This architectural topology information must be available on demand (system call or from /proc) to facilitate the necessary application partitioning early in the design stage. This approach can be taken a step further, so that an application can determine the topology dynamically and optimize its operations for the specific topology.

7.6.1.3 Priority

Low

7.6.1.4 CGL Specification

Serviceability

7.6.2 Latency API for SMP / Multi-Core Programming (CGOS-6.2)

7.6.2.1 Purpose

To support the notions of latency domain and locality domain, with the allocation of processes to latency domains and the scheduling of processes on CPUs within their locality domains. To provide APIs that allow a process to determine the locality domain characteristics of a system, including the memory latency, and also the communication latency between processes.

7.6.2.2 Description

The CGOS needs to support the allocation of processes to latency domains based on load and the scheduling of processes on CPUs within their locality domains. The CGOS also needs to also provide APIs that allow a process to determine the locality domain characteristics of a system, including the memory latency, and also the communication latency between processes.

Shared memory multi-processor systems contain multiple CPUs and memory. Memory is increasingly directly attached to CPU complexes. As a result, the cost of accessing memory attached to a CPU complex other than the one on which the application is running is more expensive than accessing local memory. Furthermore, migration of a process from one CPU to another can result in considerable cache refresh costs. As multi-core systems become more popular, these kinds of systems will become commonplace and the CGOS needs to provide support for them.

Therefore, the CGOS needs to support the notions of latency domain and locality domain. A latency domain is a set of CPUs with directly attached memory. In a system where all memory is accessed with constant cost, there is only one latency domain. In a locality domain, the virtual memory used by the processes is backed up by accessible physical memory.

The CGOS needs to be able to recognize different latency domains within the system. The CGOS needs to be able to allocate newly created processes among the latency

- Return to SCOPE, needs to be separated into multiple gaps:

1. most of this is met by NUMA, keep gap, mark it implemented by NUMA.

2. Second to last para. sounds like process affinity. Ask SCOPE for clarification. If so, create a gap based on that paragraph and mark it implemented.

3. remainder needs more definition / use cases. Suggested during working session that the remainder might be as many as three requirements?

domains based on load (current CPU consumption, memory usage). Once allocated, schedulers must attempt to maintain locality within the domain (i.e., favor CPUs in a locality domain). The CGOS needs to provide APIs that allow a process to determine the locality domain characteristics of a system. These characteristics include:

- a) The number and composition of latency domains in the system (number of CPUs, amount of memory)
- b) The domain to which a process is assigned
- c) The latency between two latency domains
- d) The allocation policy applied to an address range.

The APIs would also allow the domain to which a process is assigned to be modified and the application policy applied to an address range to be modified.

When a process connects to Unix System V shared memory or memory mapped file data shared with another process, the system might have used a round robin memory allocation policy when mapping the shared virtual memory to physical memory. Such a policy is used in the expectation that many processes will access the shared memory and the physical memory is best spread across multiple memory banks.

Some memory segments are used exclusively by a single domain, and a different memory allocation policy might be appropriate. The ability to set the memory allocation policy for a memory segment is required. The designated policy is applied when the next process touches the memory segment, assuming that the memory segment has not already been instantiated.

Besides the memory latency, the APIs must provide information on the communication latency between processes and the number of times a process has been scheduled on a different CPU.

The SCOPE Alliance recognizes that additional definition is required around the details of this gap; therefore, the Alliance welcomes the opportunity to collaborate with industry ecosystem partners and specification groups to develop additional specificity in this area.

7.6.2.3 Priority

Medium

7.6.2.4 CGL Specification

Hardware

8 Conclusion

In this document, the SCOPE Alliance CGOS Working Group has identified gaps in the Carrier Grade Linux (CGL) 4.0 requirements specification as they apply to telecommunications products and services. This document contains 16 gaps that fall into the following CGL categories:

- Availability – 1 gap
- Performance – 1 gap

- Security – 4 gaps
- Serviceability – 5 gaps
- Standards – 3 gaps
- Hardware – 2 gaps.

Moreover, it provides recommendations on the priorities for implementing these features in Carrier Grade Linux and other Carrier Grade Operating Systems. The appendix of this document contains one functional consideration (driver hardening), three non-functional considerations (application binary compatibility, application compatibility between distributions, extended support model), and one erratum (POSIX memory protection) related to the CGL 4.0 requirements specification and Carrier Grade Operating Systems that are not identified as gaps in this document.

The SCOPE Alliance CGOS Working Group has found it difficult to identify gaps for the Carrier Grade Operating System. Some gaps represent features that are not included in the CGL 4.0 requirements specification but that are already implemented in the kernel. Such features should be included in the CGL specification, and need to be validated for conformance.

LSB compliance is one of the most critical needs of the NEPs, because it reduces the costs of developing telecommunication systems and porting applications [8]. The SCOPE Alliance CGOS Working Group considers it critically important to validate conformance and to be able to configure kernel functions. When validating conformance, it must be possible to turn on configurable kernel functions and to perform additional compliance tests. It is essential that the Linux vendor perform validation testing, even if it is incremental and initially only partial, starting with performance and availability.

Given the rapid pace at which new technologies, such as new CPU architectures, are emerging, the specification and implementation of Carrier Grade Operating Systems are iterative, on-going processes. Consequently, this document is a living, on-going document, rather than the final word on this topic.

The SCOPE Alliance recognizes that more detail is required around the Trust Mechanisms, Driver Hardening, and SMP / Multi-Core Tools gaps described in this document. The Alliance welcomes the opportunity to collaborate with industry ecosystem partners and specification groups to develop additional specificity in these areas.

Through its work on identifying profiles and gaps, the SCOPE Alliance aims to accelerate the adoption of Carrier Grade Operating Systems for telecommunications equipment and applications, and to facilitate interoperability between telecommunications platforms and portability of applications.

9 Appendix

This appendix identifies one functional consideration, three non-functional considerations, and one erratum related to the CGL 4.0 requirements specification and Carrier Grade Operating Systems more generally.

9.1 Functional Considerations

This section of the appendix relates to the need for driver hardening for Carrier Grade Operating Systems.

9.1.1 Driver Hardening

9.1.1.1 Purpose

To provide support for the hardening of drivers, which includes fault containment, fault avoidance and resilience, fault and error detection, fault and error notification, logging of errors for fault diagnosis, and fault recovery.

9.1.1.2 Description

The CGOS needs to provide support for the hardening of drivers to ensure that the delivered services are available and reliable.

Memory, processors, disk drives, interface cards and the buses connecting them are being delivered with ever increasing levels of hardware integrity. Such increased levels of hardware integrity provide some degree of mitigation against the hazards of ever increasing component density.

The CGOS needs to provide interfaces and mechanisms to support the hardening of drivers, including fault containment, fault avoidance and resilience, fault and error detection, fault and error notification, logging of errors for fault diagnosis, and fault recovery.

The CGOS and the drivers must provide support for fault containment, which includes protection of the kernel from corruption by the drivers.

The drivers must be written so that they are as resilient as possible to hardware and software faults, both transient and permanent.

The CGOS and the drivers must support fault and error detection. The drivers must check for the presence of errors, particularly when accessing memory and data and, where feasible, must also verify the reasonableness of returned data.

The drivers must provide fault avoidance by recognizing incipient faults in the devices and avoiding the use of hardware that is about to fail. Tracking correctable ECC errors and retiring memory that has seen too many errors might avoid an uncorrectable error and perhaps a service outage. Extending this capability to other system components, particularly disk drives, is even more desirable.

The CGOS needs to provide programmatic/administrative interfaces that allow the drivers to report errors and to notify the users of device failures.

The CGOS needs to provide programmatic/administrative interfaces to support the logging of device errors and data errors on stable storage in order to enable subsequent fault diagnosis.

The drivers must provide support for fault recovery, including the ability to restart the drivers transparently to the applications and without damaging the data on the device.

Note: The SCOPE Alliance recognizes that additional definition is required around the details and priorities of driver hardening; therefore, the Alliance welcomes the opportunity to collaborate with industry ecosystem partners and specification groups to develop additional specificity in this area.

Also, see "Failure Resilience for Device Drivers" by J. N. Herder, H. Bos, B. Gras, P. Homburg and A. S. Tanenbaum, which won the best paper award at the 37th IEEE/IFIP International Conference on Dependable Systems and Networks [1].

9.1.1.3 Priority

High

9.1.1.4 CGL Specification

Availability

9.2 Non-Functional Considerations

This section of the appendix identifies three non-functional considerations (application binary compatibility, application compatibility between distributions, extended support model) related to Carrier Grade Operating Systems.

9.2.1 Application Binary Compatibility

9.2.1.1 Purpose

To provide application binary compatibility between releases of a distribution from the same vendor for the same CPU architecture.


9.2.1.2 Description

The CGOS needs to maintain binary compatibility between releases of a distribution from the same vendor for the same CPU architecture to enable existing applications to run unmodified on a new release.

Recompilation must not be necessary to achieve compatibility. Such compatibility means that an existing application developed on an older release of the operating system will run on the newest version unchanged, taking full advantage of the new and advanced operating system features. This requirement translates into lower development, testing, and deployment costs.

9.2.1.3 Priority

High

	<p style="text-align: center;">Carrier Grade Operating Systems</p> <p style="text-align: center;">Gap Analysis v2.0, October 23, 2007</p>
---	---

9.2.1.4 CGL Specification

Serviceability

9.2.2 Application Compatibility between Distributions

9.2.2.1 Purpose

To provide application compatibility between different distributions, from different vendors for the same CPU architecture, that comply with the same CGL specification version.

9.2.2.2 Description

The CGOS needs to provide application compatibility (both source and binary) across different distributions from different vendors, for the same CPU architecture, that comply with the same CGL specification version.

9.2.2.3 Priority

High

9.2.2.4 CGL Specification

Serviceability

9.2.3 Extended Support Model

9.2.3.1 Purpose

To ensure that the product support model for the CGOS scales to provide coverage for the lifecycle of the NEPs' products.

9.2.3.2 Description

The CGOS support model must scale to provide coverage for the duration of the lifecycle of the NEPs' products.

The NEPs have very specific product lifecycle requirements. Because the lifecycles of NEPs' solutions are long -- 12 to 36 months for development, integration and validation, 24 to 36 months for deployment, and then 5 years plus for maintenance, NEPs typically require that software and hardware vendors provide product support for an extended period of time, well beyond that of standard support policies.

The lifecycle of the CGOS needs to align to the lifecycles of the applications developed to run on it. For example, if an application is developed and released with a lifecycle of 10 years, the CGOS on which it is to be run must have a support model that provides support for the duration of the application lifecycle.

9.2.3.3 Priority

Medium

9.2.3.4 CGL Specification

Serviceability

9.3 Erratum

This section of the appendix contains an erratum for the CGL 4.0 requirements specification that apparently was unintentionally omitted from that specification.

9.3.1 POSIX Memory Protection

9.3.1.1 Purpose

To support POSIX Memory Protection (MPR).

9.3.1.2 Description

The CGOS needs to support POSIX Memory Protection (MPR).

Memory protection is a fundamental architectural characteristic of a modern operating system. It is prerequisite to build fault-tolerant systems and to achieve reliability of CG systems.

Memory protection provides a means to manage the memory access protections (read, write, execute, no-access) of any part of a process address space, with the granularity of a memory page. It is usually implemented by a combination of hardware and software.

In a shared memory environment, write restriction on data pages prevents memory data from being corrupted by a faulty process. Execution restriction prevents the introduction of new executables into a process memory space.

The CGL 4.0 requirements specification does not explicitly list POSIX Memory Protection (MPR).

The CGL 4.0 Standard requirement STD.2.3 does list MC2, which at first glance would appear to include MPR. However, the definition of MC2 is satisfied by either Memory Mapped Files (MF) or Shared Memory Objects (SHM) or Memory Protection (MPR).

Therefore, it is possible (albeit not likely) that MPR could be omitted from an implementation of the CGL 4.0 specification.

This erratum requires the CGOS to support MPR.

9.3.1.3 Priority

High

9.3.1.4 CGL Specification

Standards