# SOFTSWITCH
# EL NÚCLEO DE LAS REDES CONVERGENTES

MAURICIO ENRIQUE MIRANDA BARRIOS

DIRECTOR
ING. GONZALO DE JESÚS LÓPEZ VERGARA

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
CARTAGENA DE INDIAS D. T. Y C.
2005.

SOFTSWITCH

EL NÚCLEO DE LAS REDES CONVERGENTES

MAURICIO ENRIQUE MIRANDA BARRIOS

**Trabajo de monografía presentado como requisito para optar al titulo de ingeniero electrónico**

DIRECTORA

ING. GONZALO DE JESÚS LÓPEZ VERGARA

MAGÍSTER EN TELEMÁTICA

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

CARTAGENA DE INDIAS D. T. Y C.

2005.

Nota de aceptación

Jurado

Jurado

Cartagena D. T. Y C., Diciembre de 2005

Señores

COMITÉ CURRICULAR

UNIVERSIDAD TECNOLOGICA DE BOLIVAR

La ciudad

Respetados señores:

Con toda atención nos dirigimos a ustedes con el fin de presentarles a su consideración, estudio y aprobación la monografía titulada SOFTSWITCH, EL NÚCLEO DE LAS REDES CONVERGENTES como requisito parcial para optar al titulo de ingeniero electrónico.

Atentamente

MAURICIO MIRANDA BARRIOS.

Cartagena D. T. Y C., Diciembre de 2005

Señores

COMITÉ CURRICULAR

UNIVERSIDAD TECNOLOGICA DE BOLIVAR

La ciudad

Cordial saludo:

A través de la presente me permito entregar la monografía titulada SOFTSWITCH, EL NÚCLEO DE LAS REDES CONVERGENTES para su estudio y evaluación la cual fue realizada por el estudiante MAURICIO ENRIQUE MIRANDA BARRIOS, de la cual acepto ser su director

Atentamente,

ING. GONZALO DE JESÚS LÓPEZ VERGARA
MAGÍSTER EN TELEMÁTICA

## AUTORIZACIÓN

Yo MAURICIO ENRIQUE MIRANDA BARRIOS, identificado con la cedula de ciudadanía numero 73.205.057 de Cartagena, autorizo a la Universidad Tecnológica de Bolívar, para hacer uso de mi trabajo de monografía y publicarlo en el catalogo online de la biblioteca

MAURICIO ENRIQUE MIRANDA BARRIOS

C.C. # 73.205.057 DE CARTAGENA

## TABLA DE CONTENIDOS

## TABLA DE FIGURAS

## OBJETIVOS

*OBJETIVO GENERAL:*

- Ofrecer el soporte teórico para conocer arquitectura de la tecnología softswitch, mediante la creación de una herramienta consultiva para contribuir a la difusión del conocimiento de la misma.

*OBJETIVOS ESPECÍFICOS:*

- Generar un documento que contenga las bases teóricas de la tecnología softswitch mediante una recopilación y organización de la información esparcida por la Internet para promover el estudio de las tecnologías que dominaran el mercado en el corto plazo.

- Exponer de forma clara y concisa la migración de las redes existentes hacia redes convergentes basadas en la tecnología softswitch utilizando los conocimientos extraídos durante la investigación para hacer evidentes las ventajas que representa la implementación de la tecnología softswitch.

- Explicar y comparar los diferentes protocolos utilizados por la tecnología softswitch.

- Dar a conocer los protocolos y mecanismos propuestos por el IETF[1] y utilizados para el transporte de tráfico de audio y vídeo sobre paquetes con cierta calidad: RTP, RTCP, RSVP.

[1] *Internet Engineering Task Force*

## INTRODUCCIÓN

La infraestructura de las comunicaciones públicas conmutadas en la actualidad consiste en una variedad de diferentes redes, tecnologías y sistemas, la mayoría de las cuales se basan sobre estructuras de conmutación de circuitos. La tecnología evoluciona hacia redes basadas en paquetes y los proveedores de servicio necesitan la habilidad para interconectar sus clientes sin perder la fiabilidad, conveniencia y funcionalidad de las redes telefónicas públicas conmutadas.

La tecnología Softswitch resulta de enfocar estas necesidades. La evolución de las redes de comunicaciones públicas nos sitúa en las redes de conmutación de circuitos que predominan en la actualidad, como la red pública telefónica conmutada. Sin embargo, la próxima generación de redes nos transportará a redes convergentes basadas en paquetes como la red Internet. La idea es proporcionar una diversidad de servicios de comunicaciones basados en IP[2] equivalentes a los servicios de redes tradicionales por su calidad y facilidad de uso.

En dichas redes convergentes, actuales y futuras, se tienen que fijar las normas, y los protocolos que permitan ofrecer un rango completo de servicios de calidad sobre redes de paquetes. La definición de un estándar común es fundamental para permitir la configuración, gestión y despliegue de servicios extremo a extremo con calidad de operador sobre redes multi-vendedor y en un entorno de inter-funcionamiento con distintos operadores.

El Softswitch ofrecerá lo mejor de las redes telefónicas tradicionales e Internet, creando de esta manera un alto porcentaje de confiabilidad, combinado con rápidas reducciones en los costos e innovadores servicios. Se podrán obtener servicios y calidad similares, pero a menor precio, y se beneficiarán un porcentaje mas alto de la población por las continuas mejoras de rendimiento y costos que ofrece la tecnología de Internet. Igualmente proporcionara un

[2] Internet Protocol (Protocolo de Internet)

paquete de protocolos comunes para la interacción de las diferentes redes, posibilitando así la creación de nuevas y más poderosas redes convergentes.

La combinación de todos estos factores le permitirá a esta tecnología competir en el mismo nivel en un aspecto que las redes telefónicas tradicionales han trabajado muy bien, el QoS[3]. Las redes telefónicas tradicionales han podido ofrecer servicios de voz con niveles de calidad asegurados gracias a que cada abonado tiene un ancho de banda fijo suficiente para una transmisión de señales de voz de alta calidad; la tecnología softswitch por su parte no solo intenta proveer este mismo servicio con niveles de calidad iguales o superiores, sino además, servicios diferenciativos (video-llamadas, números personales globales, servicio de identificación de nombres, etc.) y reducción de costos debido al uso de las redes IP.

En este trabajo se explican los conceptos, normas y protocolos que se contemplan para el establecimiento de redes convergentes, se comentan, la arquitectura softswitch y las normas acordadas para comunicaciones multimedios y de VoIP[4]: H.323, SIP, MGCP[5]/MEGACO[6], así como los protocolos de transmisión RTP/RTCP, RSVP.

[3] Quality of Service (Calidad de Servicio)
[4] Voz sobre IP
[5] Media Gateway Control Protocol
[6] Media Gateway Control

# Capítulo 1
DEFINICIÓN

*Conocer la definición de lo que significa la conmutación virtual, la evolución de los medios de acceso a las redes y el impacto que tendría implantar sistemas d conmutación de paquetes en redes de voz.*

## 1. DEFINICIÓN

Softswitch es el nombre genérico para una nueva aproximación a la conmutación telefónica. Es una combinación de software y hardware como colección de productos, protocolos y aplicaciones que le facilitan a cualquier dispositivo acceso a servicios de Internet y/o telecomunicaciones en una red IP. El softswitch es donde toda la inteligencia de los servicios esta localizada para la entrega de servicios de telefonía local. Las soluciones tecnológicas del modelo softswitch pueden bajar los costos de la conmutación local y proporciona los medios para prestar servicios diferenciativos en telefonía local, con una migración muy sencilla para soportar redes de paquetes que transmitan voz de extremo a extremo.

Las redes de voz basadas en paquetes implican la digitalización, compresión y la división de la voz en paquetes. Estos paquetes pueden ser enviados del transmisor hasta el receptor por varias rutas, y luego de llegar a su destino la voz es re-ensamblada.

Por mucho, la parte más compleja de la conmutación local, es el software de control del procesamiento de la llamada. Este software tiene que tomar decisiones de enrutamiento de llamadas e implementar el procesamiento lógico de estas, para miles de características personalizadas para las diferentes llamadas. Hoy en día, los operadores locales corren estos softwares en procesadores propietarios que están fuertemente integrados con el hardware de la circuitería física como tal.

La inhabilidad de los equipos de conmutación actuales de tratar directamente con tráfico de voz en paquetes (debido a que fueron diseñados para realizar conmutación de circuitos y no se implemento un sistema para el manejo de paquetes) es la mayor barrera para la migración a la voz en paquetes. En el futuro, la telefonía local estará basada completamente en una infraestructura basada puramente en paquetes. Pero en los años siguientes, la migración hacia caminos de extremo a extremo requerirán trabajar con redes hibridas que manejen ambos tipos de redes, basadas en paquetes y en circuitos. Una posible solución a esto puede ser crear un dispositivo hibrido que pueda conmutar la voz tanto en formato de paquetes como en formato de circuitos, con todo el software de procesamiento necesario integrado en este switch.

Este enfoque puede ayudar a direccional las preguntas de migración, pero no necesariamente a bajar los costos de la conmutación o para mejorar los prospectos de servicios diferenciativos para servicios de voz locales. La industria de telecomunicaciones parece haber llegado a un consenso general, de que la mejor apuesta es separar la función del procesamiento de la llamada de la función de conmutación física, y conectar ambas mediante un protocolo estándar.

Hay varias razones por las cuales se cree que este enfoque de separación de funciones es el mejor:

- Abre la posibilidad para que surjan equipos más ágiles y pequeños, especializados en el software de procesamiento de llamadas y en el hardware de intercambio de paquetes respectivamente, lo cual tendría un impacto muy significativo en una industria que ha sido dominada por mucho tiempo por vendedores de productos integrados.

- Permite soluciones de software comunes para ser aplicadas en el procesamiento de llamadas, en un gran numero de tipos de redes; incluyendo combinaciones de redes de conmutación de circuitos y de paquetes que utilicen diferentes formatos de paquetes de voz y diferentes medios físicos de transporte.

- Permite estandarizar plataformas de computo, sistemas operativos y ambientes de desarrollo; con lo cual se tienen considerables ahorros en el desarrollo, implementación y en los aspectos de procesamiento del software de telefonía.

### *1.1 EVOLUCIÓN DEL ACCESO A LA RED.*

La aparición de tecnologías de transmisión de banda ancha basada en paquetes conocida colectivamente como DSL, esta teniendo un impacto profundo en la evolución del acceso a la red. De un punto de vista puramente de transmisión, DSL ya ha existido por algún tiempo, RDSI[7] es técnicamente hablando una variedad de DSL. Las tecnologías DSL le deben su rápida razón de crecimiento a su exclusivo uso del transporte basado en paquetes, el cual provee una solución muy efectiva en costa para aplicaciones de datos, como el acceso a Internet. VoDSL tomó ventaja de la economía de la transmisión de banda ancha en el acceso a la red y permitió una integración de los servicios de voz y datos sobre una infraestructura común. Muchas industrias han llegado a la conclusión de que el futuro de las redes de acceso telefónico reside en la implementación de redes de paquetes.

### *1.2 IMPACTO DE LOS PAQUETES DE VOZ*

Adicionalmente al impacto en las redes de acceso, donde la economía de DSL es muy llamativa, la paquetización de la voz también se ha posicionado para hacer un impacto en la red troncal, y por tanto colocando los prospectos de paquetes de voz de extremo a extremo.

La revolución de los paquetes de voz ha ido lenta, pero firmemente, ganando terreno en los últimos años. Los desarrollos iniciales de la paquetización de la voz fueron llevados a cabo por empresas utilizando redes WAN[8] de datos, para llevar el tráfico de voz de las diferentes empresas como una medida para bajar costos. El existo de este enfoque fue sin embargo, de corta duración, debido a que los proveedores de larga distancia en respuesta a esto, introdujeron precios dramáticamente más bajos en sus servicios para VPN[9] de voz, eliminando la ventaja de costo para los paquetes de voz en ese momento.

[7] Red Digital de Servicios Integrados
[8] Wide Area Network

La telefonía sobre Internet ha sido otra de las áreas claves para el desarrollo y aplicación de la paquetización de la voz. Las tarifas planas de acceso a las comunicaciones a través de Internet, ofrecen oportunidades para una reducción de costos aun mayor en las llamadas de larga distancia y llamadas internacionales. Sin embargo, el relativamente pobre desempeño de la red para dar soporte a las comunicaciones de voz en tiempo real ha limitado su éxito como una alternativa a la red publica tradicional.

A pesar de todo, el mercado ha mostrado un gran interés y sus expectativas permanecen muy altas. La razón principal para dicho interés es el incremento dramático del tráfico utilizando IP relativo al tráfico de voz en las redes de todo el mundo. El ancho de banda consumido por el trafico de voz en la red telefónica publica esta creciendo a un ritmo mucho menor que el consumo de ancho de banda para el trafico de datos en la Internet. Si el trafico de Internet continua creciendo con el mismo ritmo, dentro de muy poco, el tiempo del volumen de trafico de voz a nivel mundial, será superado por el volumen de trafico IP, mucho del cual es relacionado con Internet.

El crecimiento en el volumen del trafico IP esta llevando a una demanda de avances rápidos en las tecnologías de transmisión y conmutación de paquetes. Técnicas de DWDM[10] están exprimiendo las capacidades de multi-Gigabit de las fibras ópticas individuales, mientras que los routers núcleo de las redes están igualando este crecimiento en la capacidad de transmisión con capacidades de enrutamiento multi-Gigabit.

Como resultado de estos avances, los costos de la transmisión y la conmutación para el trafico IP están cayendo rápidamente. Por tanto es razonable pensar que los costos de la transmisión y la conmutación de la voz también caerían rápidamente si la voz fuese transportada en forma de paquetes. Ya que no hay innovaciones comparables en el mundo de la conmutación de circuitos, también se puede concluir que los costos de

[9] Virtual Private Network
[10] Dense Wavelength Division Multiplexing (Múltiplexación por División de Longitud de Onda Densa)

transmisión y conmutación de la voz no caerán tan rápidamente si esta se queda en las redes tradicionales de conmutación de circuitos.

Esta visión esta llevando a desarrollos acelerados en el campo de la paquetización de la voz. Es muy tentador concluir que en un mundo dominado por IP, la voz viajara gratis en esa infraestructura de gran capacidad. Pero los requerimientos para el transporte de la voz son fundamentalmente diferentes a los de datos; y no es muy claro en este momento como una solución basada puramente en IP pueda proveer una calidad de servicio en canales de paquetes de voz con tiempos aceptables de transmisión de extremo a extremo en una escala global.

Esta es la mayor preocupación de los proveedores de servicios, debido a que el tráfico de voz es el que sigue dominando sus ingresos y sus márgenes de ganancia. Como resultado, mucho del trafico IP en redes de gran capacidad esta siendo transportado sobre redes ATM[11] debido a que estas ofrecen capacidades para el transporte de paquetes para voz y datos junto con niveles de garantizados de QoS, que son tan importantes para la voz.

El caso económico para aplicar técnicas de conmutación basadas en paquetes a la voz es muy claro. Lo que es menos claro es como las técnicas de conmutación de paquetes dominaran el mundo de la voz en el largo plazo. Los proponentes de voz basada en IP proponen la eliminación total de la capa ATM, pero hasta que IP demuestre claramente su habilidad para proveer de forma escalable y económica, soluciones atractivas que protejan la calidad de la voz en las transmisiones, la apuesta segura seguirá siendo ATM para aquellos que ya tengan esta tecnología instalada o MPLS[12] para los nuevos jugadores.

### *1.3 RETOS PARA LOS PROVEEDORES LOCALES*

[11] Asynchronous Transfer Mode (Modo de Transferencia Asíncrono)
[12] Multiprotocol Label Switching

Empresas, clientes y negocios de todos los tamaños disfrutan los beneficios del actual mercado de servicios de telecomunicaciones actual, debido a los altos niveles de competencia. El volumen de trafico generado por estos clientes justifica el uso de circuitos dedicados para necesidades de ciertos clientes en redes locales y de larga distancia; y los PBX utilizados por estos clientes proveen inteligencia para el enrutamiento de llamadas que dirige el trafico, llamada por llamada, hacia el proveedor mas adecuado.

Los servicios telefónicos contratados por estos clientes se relacionan típicamente con el transporte básico de voz entre varias localizaciones de la empresa, y la conexión del tráfico de voz con varias redes publicas. Los PYMES[13] y los clientes residenciales, no tienen la ventaja de una oferta de múltiples servicios en competencia, las condiciones económicas de estos clientes no soportan conexiones directas hacia las redes de larga distancia, por lo tanto son dependientes de un único servicio para la telefonía. Estos segmentos de mercado generan la masa de los ingresos para los proveedores locales.

Los proveedores de servicios tienen grandes cantidades de ingresos provenientes de las llamadas por minuto, del establecimiento y la liberación de llamadas de larga distancia y del uso de características como el correo de voz. Como no ha habido una competencia efectiva en este sector del mercado, las tarifas se han mantenido relativamente altas, con altos márgenes de ganancia. Hoy en día, todos los servicios locales de voz son derivados de los switchs tradicionales basados en circuitos, más que nada porque no existen otras soluciones. Esto representa una gran barrera para los servicios por las siguientes tres razones clave:

1. Costo de las soluciones de conmutación local.
2. Falta de diferenciación de servicios.
3. Barreras para la migración de redes.

[13] Pequeñas y medianas empresas.

### 1.3.1 COSTO DE LAS SOLUCIONES DE CONMUTACIÓN LOCAL.

El mercado de la conmutación local esta dominado actualmente por unos pocos y poderosos actores que han construido negocios con ganancias muy altas, sirviendo solo a las necesidades de los proveedores locales. Los switchs de conmutación local de estos vendedores están optimizados para recibir cientos de miles de líneas.

Mientas que la escalabilidad de estos dispositivos no se pone en duda, el alto costo de los equipos que comúnmente son utilizados le impide a los proveedores enfocarse a mercados pequeños y apuntar sus servicios hacia los mercados más grandes. Si los costos de las soluciones de conmutación local fuesen menores, la competencia entre los proveedores locales se vería estimulada y los clientes se beneficiarían debido a que podrían elegir el proveedor con las tarifas mas bajas.

### 1.3.2 FALTA DE DIFERENCIACIÓN DE SERVICIOS.

Todos los switchs de conmutación local ofrecen el mismo conjunto de características para servicios de llamadas específicos; llamada en espera, desvío de llamadas, identificador de llamadas, etc. La mayoría de estas características han estado disponibles por muchos años y la aparición de nuevas características es muy rara. Esto se debe principalmente a que es muy costoso desarrollar y probar nuevas características, o implementarlas sin comprometer la estabilidad de las actualmente existentes.

Como los proveedores son dependientes hoy día dependen de los mismos juegos de productos para la conmutación local, se encuentran limitados a ofrecer exactamente los mismos servicios. En estas circunstancias, la única forma para un proveedor de servicios competitivo de ganar clientes es bajar sus tarifas. La diferenciación basada solamente en el precio ha probado no ser una buena estrategia de negocios a largo plazo en el sector de las telecomunicaciones. Si las soluciones de conmutación tuviesen disponibles características realmente nuevas y atractivas para ofrecer, entonces los proveedores tendrían la oportunidad de diferenciar sus servicios en aspectos diferentes al precio, esto podría ofrecer perspectivas mucho más claras para las ganancias y la retención de clientes.

### 1.3.3 BARRERAS PARA LA MIGRACIÓN DE REDES.

Los switchs para conmutación tradicionales están basados en técnicas de conmutación de circuitos. Dentro de la fabricación del switch, el tráfico de voz esta representado como ráfagas de 64kbps, en las entadas y salidas de switch, las ráfagas de 64kbps son multiplexadas por división de tiempo dentro de ráfagas de mayor velocidad.

La inteligencia del switch que realiza las funciones de enrutamiento de las llamadas y el procesamiento de las características, esta integrado íntimamente con la fabricación de la conmutación de circuitos. Las ventajas económicas de la paquetización de la voz están dirigiendo el acceso y el núcleo de las redes de voz lejos de la conmutación de circuitos y cada vez más cerca de la conmutación de paquetes.

Como los paquetes de voz han ido siendo aceptados ampliamente tanto en la red de acceso como en el núcleo de las redes, los switchs convencionales se convierten en islas que conectan dos redes de paquetes y la conversión de paquetes a circuitos debe ser llevada a cabo en la entrada y en la salida del switch, sin embargo, esto introduce costos y retardos de transmisión indeseables para el camino de la voz.

Si una solución de conmutación local tuviese la capacidad de entregar servicios de voz locales, y características personalizables directamente sobre una infraestructura de conmutación de paquetes, la conversión de paquetes a circuitos seria innecesaria y por tanto podría eliminarse. Esto tiene el doble efecto de reducir costos y mejorar la calidad, además mueve la red de voz un paso más cerca de la meta ultima, una red homogénea de paquetes de voz de extremo a extremo.

### *1.4 TRÁFICO MULTIMEDIOS. EL DESAFÍO DEL TIEMPO REAL*

El despliegue y utilización de las redes multimedios se encuentra con una serie de dificultades prácticas cuya resolución no es trivial. En efecto, el tráfico multimedia, caracterizado por su densidad y carga de datos, supone un desafío para su funcionamiento operativo en Internet.

En primer lugar esta la necesidad de capacidad (ancho de banda) para las aplicaciones de audio-video que no son comparables a las necesarias para la transmisión de texto, gráficos o imágenes. Además, el tráfico multimedia es de tiempo real. A diferencia del tráfico de datos, donde la importancia de la velocidad es relativa, una latencia (retardo total) de más de 250ms contribuye a la degradación de las transmisiones de voz o de imágenes, y el usuario se queja entonces de la calidad. Si la red además se congestiona y hay paquetes que no llegan a su destino, la retransmisión de paquetes perdidos agrava la situación aún más.

Por otro lado los datos multimedios se transmiten por lo general a ráfagas, a veces van a distintos puntos a la vez y no son predecibles. Las memorias utilizadas para compensar los efectos de velocidad variable son limitadas, y pueden sufrir sobrecarga de datos, si la velocidad de paquetes es muy alta a la entrada, o bien de paros momentáneos de transmisión si los paquetes van demasiado lentos. Frente a ello hay que tener en cuenta que las redes en servicio pueden estar compartidas por miles de usuarios, y poseen una capacidad, disponibilidad y retardo limitado o impredecible.

Las redes dedicadas utilizando Gigabit Ethernet, FDDI y ATM proporcionan los anchos de banda necesarios para audio y video digital. ATM en particular podría ser una solución para la transmisión de vídeo a alta velocidad debido a su gran capacidad de transmisión, por ser una tecnología orientada a conexión y con posibilidad de ofrecer calidad de servicio diferenciada según las aplicaciones. En este momento sin embargo, las redes ATM se emplean principalmente en redes WAN, troncales o de acceso con ADSL, y normalmente no llegan al usuario final.

# **Capítulo 2**
# Modelo de Arquitectura Softswitch

*Conocer los componentes estructurales y los requisitos funcionales de cada sección del softswitch; además de una breve introducción a los protocolos utilizados para la intercomunicación entre los mismos*

## 2. MODELO DE ARQUITECTURA SOFTSWITCH

Softswitch es un nuevo concepto de conmutación de telefonía por paquetes que trata de solventar los inconvenientes de las centrales de conmutación mediante la separación del hardware y el software de red.

En las redes de conmutación tradicionales, las aplicaciones que controlan las llamadas (en su gran mayoría, aplicaciones propietarias) deben tomar muchas decisiones de encaminamiento en milisegundos, y los elementos que componen su lógica de control, están totalmente integrados en la circuitería de conmutación física y constituyen la parte más compleja de una central de conmutación.

Por otro lado, con vistas a una convergencia de voz y datos, la migración hacia una infraestructura de paquetes extremo a extremo requiere la instalación de una red híbrida, que maneje tanto conmutación de circuitos como de paquetes y de manera transparente al usuario. Con el objetivo de llegar a dicha convergencia se formó el ISC[14], que es un consorcio de la industria de las telecomunicaciones que, como el IETF[15], elaboran acuerdos para llegar a niveles de conmutación que separen las funciones de control de las funciones de conmutación física (conmutación virtual).

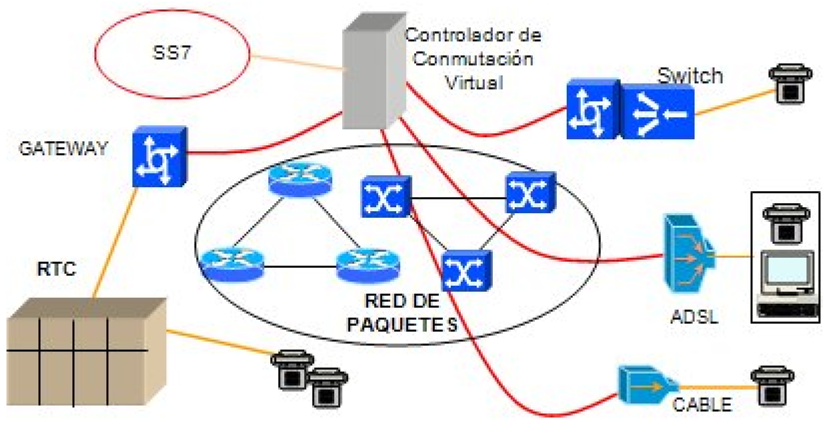


**Figura 1 Arquitectura de una red de conmutación virtual**

---

[14] *International Softswitch Consortium*
[15] *Internet Engineering Task Force*

En la figura 1 se puede apreciar la arquitectura de una red cuyo sistema de conmutación sea virtual. Se debe tener un elemento de control central (Control de conmutación virtual) que se encargue de recibir las señales de las redes que se deseen hacer converger, el cual deberá contar con los medios adecuados para codificar y decodificar las señales provenientes de cada una de esas redes, así como los medios para establecer puentes entre cada una de ellas.

En la misma figura se ilustra como utilizar una red de paquetes (red IP) para ofrecer servicios de transmisión de voz mediante diferentes medios. Para los teléfonos que posean el protocolo IP nativo se conectaran directamente al conmutador virtual utilizando la red IP, mientras que los teléfonos tradicionales utilizarían las redes convencionales, las cuales se conectan al conmutador virtual mediante un equipo traductor de medios (un gateway) y así se interconectan las redes involucradas.

La tecnología de conmutación de un softswitch reside en el software (de ahí su nombre) en lugar del hardware como sucede con la tecnología tradicional de centrales de conmutación. Esta característica le da un arma muy importante a esta tecnología, la programabildad, la cual le permite dar soporte a diversos protocolos y versiones de telefonía IP (H.323, SIP, MEGACO, etc.).

Un softswitch debe ser capaz de realizar las funciones que realizan las centrales de conmutación actuales, con capacidades adicionales para dar un manejo eficiente a la conmutación de paquetes, debe contar con servicios de conexión que funcionen como puentes entre los diferentes de medios y/o un terminal IP nativo, seleccionar procesos a aplicar a una llamada, proveer encaminamiento a una llamada basada en la señalización e información del cliente, transferir el control de la llamada a otro elemento de red y servir de interfase a diferentes funciones de soporte de gestión de servicios.

Entre las ventajas que proporciona la separación de funciones del softswitch se destacan:

1. Escalabilidad y más rapidez en el despliegue de red.
2. Flexibilidad en la operación: provisión, mantenimiento, gestión y supervisión de red.
3. Permitir una solución común para el proceso de las llamadas, que se puede aplicar en conmutación de circuitos y de paquetes, independientemente del formato y del transporte físico (multiprotocolo).
4. Posibilidad de tener la inteligencia centralizada, basada en plataformas y sistemas operativos abiertos, con el consiguiente ahorro en los procesos de desarrollo e implantación de las aplicaciones de telefonía.

Los esfuerzos del ISC y del IETF se han centrado en establecer una arquitectura distribuida, con componentes funcionalmente independientes: transporte, conmutación, control y lógica del servicio. La ventaja que ofrece una lógica distribuida es que las aplicaciones pueden crear, controlar o entregar diferentes tipos de servicios desde diferentes sitios de la red, sin necesidad de que los servicios estén centralizados, lo cual le entrega una mayor flexibilidad a todo el sistema. La fig. 2 muestra un esquema de conmutador de telefonía IP distribuido con sus módulos funcionales lógicamente separados. Las pasarelas (gateways) tendrían contacto directo con los elementos de la red (PTS[16] y PCS[17]) mientras que la parte lógica tomaría las decisiones de enrutamiento de llamadas y llevaría el control de las mismas.

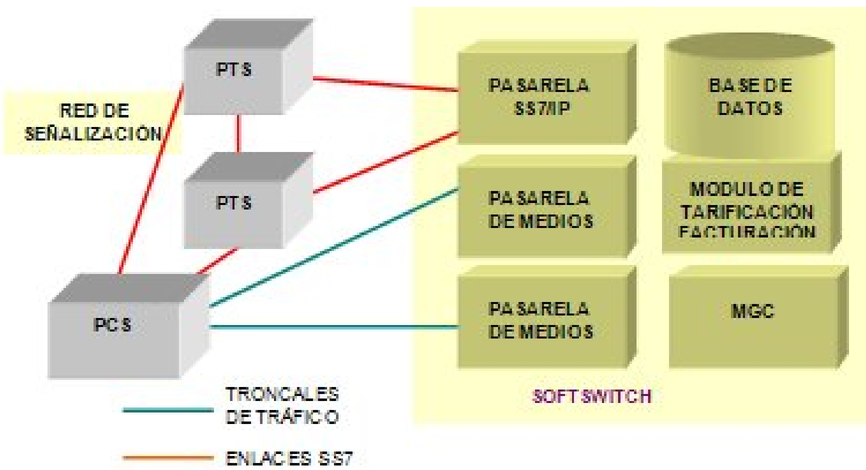


**Figura 2 Conmutación IP distribuida**

[16] Signaling Point (Punto de Señalización)

### *2.1 COMPONENTES SOFTSWITCH*

Los componentes que forman parte del concepto softswitch son:

- El gateway de (gestión de) medios (MG)**[18]**, que es el elemento que realiza la conmutación física,
- El gateway de control de medios (MGC)**[19]**, realiza el control y tratamiento de llamadas e intercambio de mensajes de señalización.
- El gateway de señalización (SG)[20] transforma la señalización de red conmutada en señalización por red IP.

#### 2.1.1 GATEWAY DE GESTIÓN DE MEDIOS (MC)

El MG puede ser considerado como el hardware de conmutación del softswitch, es elemento encargado de interconectar físicamente las diferentes redes que se conecten a el. Este elemento toma sus directrices del MGC, el cual le indica como, cuando y con que recursos interconectar las redes. El MG es el encargado de proporcionar un medio para transportar voz, datos, fax y vídeo entre la red de paquetes IP y una RTPC[21]. Es decir, controla la conversión de la voz en paquetes para la red IP; para transportar estos datos a una red de paquetes la voz muestreada debe ser comprimida y paquetizada.

Los DSP[22] realizan las funciones de conversión análoga a digital, compresión de voz a código de audio, cancelación de eco, detección de silencios, supresión, compresión de código, generación de ruido aceptable y transporte de señal DTMF fuera de banda, etc., entre otras funciones

El MG debe soportar los siguientes requisitos funcionales:

---

[17] Commutation Point (Punto de Conmutación)
[18] *Media Gateway*
[19] *Media Getaway Controller*
[20] *Signaling Gateway*
[21] Red Telefónica Publica Conmutada
[22] Digital Signal Processors (Procesadores de Señal Digital)

- Transmisión de la voz como datos utilizando el protocolo de transmisión RTP (Real-time Transmission Protocol)
- Recurso suficientes en DSP's y gestión de estos recursos para proporcionar voz y funciones de paquetes para los servicios mencionados.
- Asignación de Intervalos de Tiempo E1 bajo el control del MGC como un resultado de mensajes MCGP, MEGACO o SIP.
- Soportar configuraciones tipo "clear channels" E1 para tráfico de voz en redes SS7.
- Gestión de recursos y enlaces E1.
- Tarjetas y conectores DSP y E1.
- Facilidad de escalamiento incluyendo puertos, tarjetas y nodos sin impactar otros componentes del switch.

Para la intercomunicación entre el MG y el MGC, actualmente las soluciones que se contemplan se basan en el protocolo **MGCP** y la ampliación a **MEGACO** (UIT H.248). El protocolo MEGACO proporciona la capacidad de manejar diversos tipos de flujos de medios bajo el control del MGC, incluyendo la señalización asociada a esos medios. MEGACO permite a los gateways "dialogar" con los sistemas de señalización de las redes conmutadas.

La figura 3 ilustra como es el funcionamiento de una arquitectura softswitch; tanto las conexiones provenientes de la red IP como de la RTPC tienen una conexión directa con el MG, el cual coordina las funciones de conmutación con el MGC, siendo este ultimo quien establece las instrucciones de conmutación, realiza el control de tonos, etc.

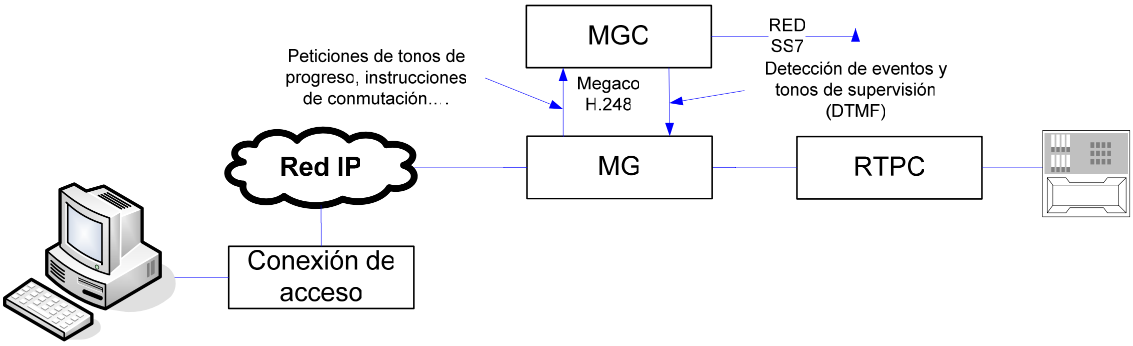


**Figura 3 Arquitectura Softswitch**

MEGACO/H.248 es un protocolo extensible, e incluye mecanismos para adoptar y registrar nuevos paquetes de aplicación para necesidades diversas, como variaciones nacionales de servicios telefónicos (una explicación mas detallada de este protocolo se encuentra en la siguiente sección).

Una solución softswitch completa debe soportar el inter-funcionamiento de conexiones de diversa índole, tales como accesos basados en voz sobre paquetes, conexiones troncales de circuitos digitales entre centrales de tránsito o circuitos de voz sobre paquetes. Igualmente debe proveer los medios para poder manejar conexiones de datos y video para dar soporte al llamado “Triple Play” de las telecomunicaciones, voz – datos – video sobre una misma conexión.

Cada combinación de conexión en la red de acceso y red troncal requiere unas capacidades del gateway de medios para soportar el acceso telefónico local. Las funciones del MG pueden ser por tanto desde la simple conmutación, como en una central telefónica, hasta la conversión de redes troncales de conmutación de circuitos en paquetes, y pueden estar localizadas en dependencias del usuario o del proveedor de servicios.

La figura 4 muestra un ejemplo de implantación de softswitch sobre una arquitectura de red de paquetes desde distintas redes de acceso (se ha omitido el gateway de control de medios para mayor claridad). Se ilustra en que lugar de la red de acceso debe ser implantado el MG para poder funcionar de forma correcta. El MG es el elemento en color oscuro y es quien proporciona el tono de llamada y realiza las funciones de una central local; en el ejemplo se muestra como un MG puede servir como elemento de acceso para clientes con redes analógicas o digitales de forma transparente (gracias a su capacidad para recibir diferentes tipos de medios) y a su vez realizar la transmisión de la información por cualquiera de estos medios.

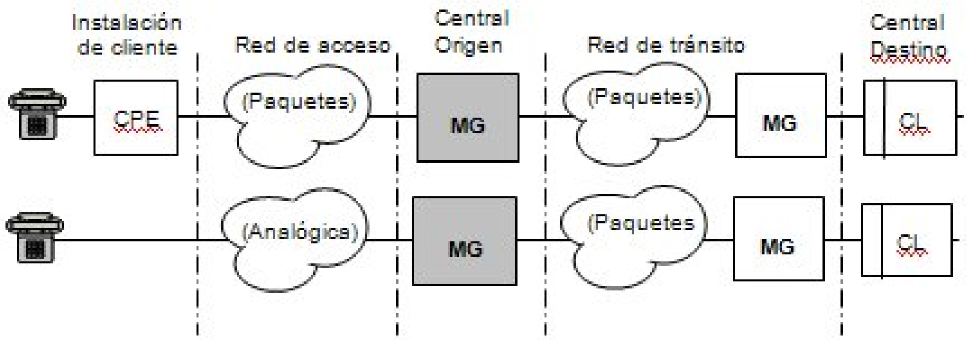


**Figura 4 Ejemplo de implantación de MG's sobre redes de paquetes**

### 2.1.2 GATEWAY DE CONTROL DE MEDIOS (MGC)

Los controladores de medios transfieren la inteligencia de la conmutación hacia una base de datos o servidor de aplicaciones, proporcionando el control de las redes de próxima generación (NGN[23]). El MGC debe disponer de ciertas funcionalidades que se distribuyen según tres niveles de complejidad creciente: agentes de llamada, servicios básicos y entorno de creación de servicios.

Los agentes de llamada constituyen la funcionalidad básica del MGC, y sirven para establecer y terminar las llamadas, y mantener detalles del estado de las mismas. La funcionalidad incluye el enrutamiento de las llamadas como en las centrales locales, y puede resultar más complejo en el caso de soportar traducción de números y servicios de red inteligente. Los agentes de llamadas interactúan con los protocolos de señalización que existen en cualquiera de los dos extremos con el propósito de coordinar el establecimiento y la finalización de las llamadas. Por ejemplo, un MG que soporte la conversión de señales SS7 a conexiones VoIP requiere un MGC con un agente de llamadas que pueda manejar mensajes SS7 y los mensajes de control de llamada de H.245, el cual le da soporte al establecimiento de llamadas VoIP como parte del paquete de protocolos H.323.

Un MGC que contiene solo la funcionalidad de agente de llamadas puede proveer servicios de telefonía local a clientes que usan PBX debido a que el PBX tiene muchos servicios incluidos para el usuario final y la conexión del PBX con la red publica requiere solo servicios básicos de transporte, los cuales pueden ser suministrados por un agente de llamadas. Los negocios pequeños y los clientes residenciales sin embargo, no tienen (generalmente) acceso a un PBX y esperan que ciertas funciones sean proporcionadas por la red. Aquí entran en juego los MGC que contienen servicios básicos, algunos de los servicios más comunes para la telefonía local son:

- Llamada en espera.
- Desvío total de llamadas.
- Desvío de llamadas cuando la línea este ocupada.
- Desvío de llamadas al no responder.
- Desvío de llamada selectivo.
- Activación remota del desvío de llamadas.
- Transferencia de llamadas.
- Conferencia de llamadas.
- Timbre personalizado.
- No molestar.
- Identificación  de llamadas.
- Bloqueo de llamadas entrantes.
- Llamada en espera para llamadas entrantes.
- Rediscado automático.
- Rechazo selectivo de llamadas.
- Restricciones para la línea.
- Marcación con un solo digito.

[23] New Generation Networks

Un MGC que solo implemente un agente de llamadas y un juego básico de características para llamadas, puede proveer una solución para telecomunicación local basada en softswitch para telefonía local que ofrezca una funcionalidad comparable a los servicios de telecomunicaciones locales (tal vez a un costo mucho menor) pero no proveería ningún medio para crear servicios diferenciativos. El entorno de creación de servicios es un elemento clave para atraer y mantener clientes y por tanto un MGC para servicios de telefonía local debe tener la habilidad para crear y personalizar los servicios del proveedor. Las centrales telefónicas actuales no ofrecen esta clase de opciones, todas las características de conmutación son implementadas dentro de su software y no existen interfaces públicas en las cuales se puedan agregar o modificar características de conmutación.

Algunas características especiales pueden ser implementadas por fuera del switch utilizando interfaces de señalización estándar, como SS7, y esto es la base para el concepto de las redes avanzadas inteligentes (AIN[24]). Pero las AIN han prometido mas de lo que pueden ofrecer, esto es debido a que muchas características deseables requieren interacción directa con la maquina de estados de la llamada, y las AIN no tienen esa capacidad. Desarrollar nuevas características en una central telefónica tradicional, requiere abrir el software incrustado y escribir el código en las interfaces de programación internas (API[25]) del switch.

En muchos de los switchs actuales, el código base ha evolucionado por muchos años, con una creciente complejidad de dicho código, lo cual garantiza la complejidad para el desarrollo de nuevas características; lo cual conlleva a que se deba revisar exhaustivamente cualquier adición para no introducir errores en ninguna de las características ya existentes. Si los proveedores de servicios tienen la oportunidad de crear sus nuevas características, se requiere un enfoque distinto para la arquitectura de procesamiento del MGC:

[24] Advanced Intelligent Network
[25] Application Programming Interfaces

- Un lenguaje de alto nivel se requiere para definir la funcionalidad de nuevas características, y debe estar ligado a herramientas de desarrollo graficas, que permitan que estas características sean diseñadas visualmente, sin complejas reglas de código.
- Se requiere un enfoque completamente orientado a objetos, en donde las primitivas básicas son implementadas como objetos y nuevas características se pueden construir basadas en esas primitivas y tomando ventaja de su herencia (los objetos implementados sobre primitivas básicas tienen todas las capacidades de dichas primitivas junto con las nuevas que desarrollen dichos objetos).
- El modelo de objetos para el procesamiento de llamadas debe tener en cuenta posibles interacciones entre primitivas básicas, haciendo innecesarias las pruebas de características nuevas desarrolladas en base a dichas primitivas.
- Los datos que describen la configuración de las características de cada subscriptor y los parámetros que las controlan, deben ser accesibles a través de un formato de intercambio de datos que facilite el desarrollo vía Web.

Un MGC que tome este enfoque y ofrezca un ambiente de creación de características robusto y fácil de usar, les da un gran poder a los proveedores de servicios de telefonía. El bajo costo y el corto tiempo de desarrollo para las nuevas características significa que por primera vez, los proveedores de servicios pueden crear servicios especializados para segmentos específicos del mercado, en lugar de utilizar características genéricas que son utilizadas en los switchs actuales.

Un MGC puede controlar uno o varios MGs, dependiendo de la capacidad de proceso de llamadas. También deben ser capaces de soportar los protocolos de señalización asociados a las redes que se conectan a los MGs: H.245, SIP, etc., (redes de paquetes) o bien SS7/ISUP, CAS (redes de circuitos). Por ser IP el modo nativo de comunicación del MGC con el exterior, es necesario un método para que los protocolos de señalización basados en circuitos se conviertan en protocolos basados en transporte IP. La solución natural es utilizar el protocolo SCTP de SIGTRAN,

Para finalizar, se debe especificar que los MGC deben soportar las siguientes características funcionales:

- Establecimiento del control de la llamada.
- Protocolos de establecimiento de las llamadas de voz: H.323, SIP.
- Protocolos de control de medios: MGCP, MEGACO H.248.
- Clases de Servicios y Control de Calidad del Servicio.
- Protocolo de Control: SIGTRAN (SS7 sobre IP)
- Procesamiento SS7.
- QoS relacionada con el manejo de mensajes de protocolo.
- Enrutamiento, que incluye.
  - Componentes de enrutamiento: Plan de marcación local (E164)
  - Superposición de análisis de dígitos y/o señalización dentro de bloque
  - Soporte de traducción de dígitos para FR, IP, ATM y otra redes
- Registro de detalles de llamadas para facturación
- Control de gestión de la anchura de banda
- Aprovisionamiento para pasarelas de medios:
  - Asignación de canales de 64 Kbps
  - Transmisión de la voz (codificación, compresión y paquetización) y otros
- Aprovisionamiento para pasarelas de señalización
  - Variantes SS7
  - Temporizadores de procesos y otros
- Registro del Gatekeeper (identificación, supervisión, control del tráfico pasante)

### 2.1.3 GATEWAY DE SEÑALIZACIÓN (SG)

El SG se encarga de realizar la traducción entre el protocolo SS7 y el protocolo IP. Recibe e interpreta la información de señalización y le informa esto al MGC para que este le indique al MG como realizar la conmutación de la información. Su función principal es la de crear un puente entre la red SS7 y una red IP, bajo el control del MGC y hacer ver al softswitch como un nodo normal SS7 (Punto de Señalización) en una red SS7.

Debe soportar las siguientes funciones:

- Proporcionar conectividad física a la red SS7 vía conexiones físicas T1/E1 o T1/V.35
- Estar en capacidad de transportar información SS7 entre el controlador de la pasarela y la pasarela de señalización vía una red IP.
- Proporcionar una trayectoria de transmisión para voz, vídeo y opcionalmente datos.
- Proporcionar una operación SS7 altamente disponible para servicios de telecomunicaciones.

El protocolo de señalización SCTP (quien realiza la traducción entre SS7 e IP) y su funcionamiento se explica en la próxima sección.

### *2.2 PROTOCOLO SCTP. SIGTRAN*

El grupo de trabajo SIGTRAN *(Signaling Transport)* del IETF ha propuesto el protocolo de control SCTP (*Stream Control Transmision Protocol*) para el transporte de señalización de redes públicas tradicionales basada en paquetes sobre redes IP. Originalmente SCTP fue diseñado para proveer un protocolo de transporte de propósito general para aplicaciones orientadas a mensajes, ya que se necesitaba para la transmisión de datos de señalización. Su diseño incluye comportamientos apropiados para evitar las congestiones y resistencia a ataques de negación de servicio y de suplantación.

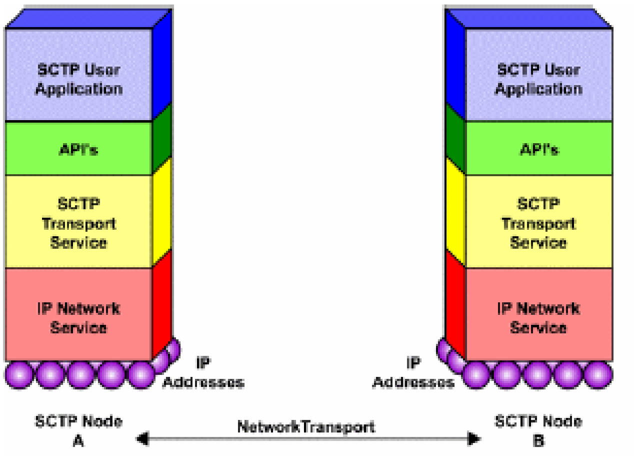


**Figura 5 Protocolo SCTP**

SCTP puede sustituir con ventaja a UDP y TCP en el transporte de aplicaciones transaccionales, particularmente las de señalización, lo que le hace apropiado para el transporte de mensajes de señalización como Q.931 o SS7 PUSI[26]. La diferencia decisiva con TCP es el "multhoming" (que será explicado más adelante) y el concepto de varios canales dentro de una misma conexión (también conocido como asociación). Mientras que en TCP un canal es conocido como una secuencia de bytes, en SCTP un canal es una secuencia de mensajes (y estos pueden variar en longitud). SCTP puede ser usada como un protocolo de transporte cuando se requiere monitorear y detectar la pérdida de una sesión; para tales aplicaciones, los mecanismos de detección de fallas en la trayectoria o en la sesión de SCTP monitorearan activamente la conectividad de la sesión.

SCTP incluye además, mecanismos de control de congestión, validación de mensajes y gestión de encaminamiento concebidos para el transporte de la señalización con características mejoradas. Así, una asociación SCTP (que viene a ser equivalente a una conexión TCP) puede contener varios "canales" (*streams*) lógicos independientes de datos, cada uno con su propio control de flujo.

### 2.2.1 PAQUETES SCTP.

[26] *Parte de Usuario de Servicios Integrados*

Las PDU[27] de SCTP son llamados paquetes SCTP, si SCTP corre sobre IP, un paquete SCTP forma el payload del paquete IP. Un paquete SCTP esta compuesto de una cabecera común y varios *chunks.* Varios *chunks* pueden ser multiplexados dentro de un solo paquete que tenga en tamaño máximo igual al del MTU[28] de la trayectoria que debe recorrer para llegar a su destino. Un *chunk* puede contener información de control o datos de usuario.

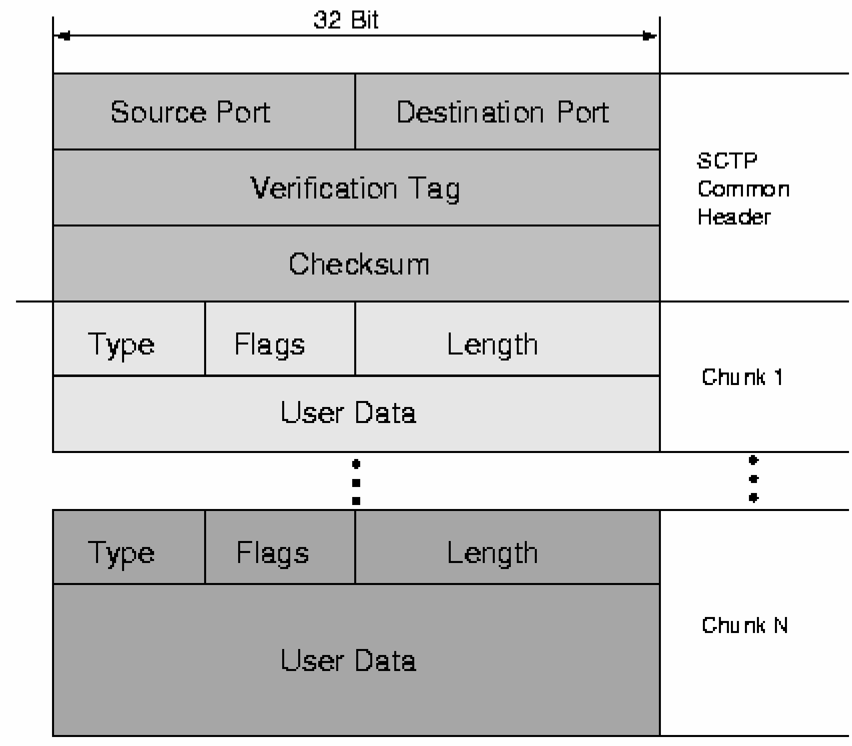


**Figura 6 PDU de SCTP con varios chunks.**

La figura 7 muestra la estructura de un paquete SCTP. La cabecera común consiste de 12 bytes. Para la identificación de una asociación, SCTP utiliza el mismo concepto de puertos que TCP y UDP. Para la detección de errores de transmisión, cada paquete SCTP esta protegido con un *checksum* de 32 bits (Utilizan el algoritmo Adler-32), el cual es mucho más robusto que el *checksum* de 16 bits de TCP y UDP. Los paquetes SCTP con un *checksum* inválido son descartados. La cabecera común también contiene un valor de 32 bits llamado *verification tag* (bandera de verificación) la cual es especifica para cada asociación y es definida e intercambiada entre los puntos extremos de transmisión al establecerse la asociación y por tanto existen dos banderas de verificación en cada asociación.

[27] Protocol Data Unit (Unidad de Datos del Protocolo)
[28] Maximum Transmission Unit

Cada *chunk* comienza con un campo que indica de que tipo es, este campo es utilizado para distinguir *chunks* de datos de *chunks* de control. Luego se encuentran banderas específicas de cada *chunk* y el tamaño del mismo (este campo se hace necesario debido al tamaño variable de los *chunks*). El campo de valor contiene el *payload* real del *chunk*. Hasta ahora hay 13 tipos de *chunks* definidos para uso estándar. Su listado y sus definiciones se pueden encontrar en el RFC2960.

### 2.2.2 TRANSMISIÓN DE DATOS EN SCTP

Las implementaciones de SCTP deben tener mecanismos de control de flujo y de congestiones de acuerdo con el RFC2960, en el que se asegura que SCTP puede ser introducido sin problemas en redes donde TCP es ampliamente usado.

SCTP opera en dos niveles:

- Dentro de una asociación, la transferencia segura de datagramas es asegurada usando un *checksum*, un número de secuencia y un mecanismo selectivo de retransmisión. Sin tomar la secuencia inicial en consideración, cada *chunk* de datos recibido correctamente es entregado a un segundo nivel, completamente independiente del primero.
- El segundo nivel realiza un mecanismo flexible de entrega, el cual esta basado en la noción de varios canales dentro de una asociación. Los *chunks* que pertenecen a uno o varios canales pueden ser ligados y transmitidos en un paquete SCTP no mayor que el MTU de la trayectoria actual.

La detección de pérdidas y duplicados de *chunks* de datos esta habilitada gracias a la numeración de todos los *chunks* de datos en el transmisor con el

TSN[29]. Los *acknowledgements* (reconocimientos) enviados desde el receptor al transmisor están basados en estos números de secuencia.

Las retransmisiones son controladas con base en el tiempo. La duración del temporizador se deriva de mediciones continuas del retardo de transmisión. En el momento que un temporizador de retransmisión expira (y los controles de congestión permiten la transmisión) todos los chunks de datos no reconocidos son retransmitidos y el temporizador es duplicado respecto de su duración inicial (como en TCP). Cuando el receptor detecta uno o más saltos en la secuencia de los chunks de datos, cada paquete SCTP recibido es reconocido mediante el envío de un SACK[30] que reporta todos los saltos. Cada SACK esta contenido en un chunk de control especifico. Cuando el transmisor recibe cuatro SACKs consecutivos reportando los mismos chunks de datos perdidos, estos son inmediatamente retransmitidos (retransmisión rápida). La mayoría de los sistemas actualizados ya soportan extensiones similares para TCP (RFC 2018).

### 2.2.3 CONTROL DE FLUJO

SCTP utiliza un mecanismo de control de congestión y de flujo basado en una ventana de extremo a extremo similar a la utilizada en TCP. El receptor puede controlar la tasa a la que el transmisor esta enviando, especificando un tamaño de ventana basado en octetos y devolviendo este valor junto con todos los SACKs *chunks*. El transmisor mantiene una variable conocida como CWND[31] que controla el número máximo de bytes que pueden ser enviados antes de que sean reconocidos. Cada *chunk* de datos debe ser reconocido y el receptor debe esperar cierto tiempo (usualmente 200ms) antes de que esto se haga.

*Control de flujo para los puntos finales multihomed.*

Por defecto, toda transmisión se dirige hacia una dirección previamente seleccionada del juego de direcciones de destino, que se conoce como dirección primaria. Las retransmisiones se deben hacer en diversas

[29] Transport Sequence Number (Numero de Secuencia de Transporte)
[30] Selective Acknowledgement

trayectorias, de modo que si se sobrecarga una trayectoria, las retransmisiones no afecten esta trayectoria (a menos que la topología de la red sea tal que las retransmisiones lleguen al mismo punto en la red donde estaban los datos perdidos debido a la congestión).

Si la trayectoria activa tiene un alto numero de faltas y su contador de error excede un limite, SCTP notifica al proceso de capa superior que la trayectoria ha llegado a ser inactiva. Entonces una trayectoria primaria nueva se puede (y debe probablemente) elegir para su uso.

*Control de la congestión*

El comportamiento del control de la congestión de SCTP se define en el RFC2960 y puede tener un impacto donde la entrega oportuna de mensajes sea muy importante (por ejemplo, el transporte de datos de señalización). Igualmente asegura el comportamiento apropiado de SCTP cuando se introduce en gran escala en redes de intercambio de paquetes existente tales como el Internet. Los mecanismos del control de la congestión para SCTP se han derivado del RFC 2581 (control de flujo para TCP), y se han adaptado para multihoming. Para cada dirección de destino (es decir cada trayectoria posible) un sistema discreto de parámetros del control de flujo y del control de la congestión se guarda, de forma que desde el punto de vista de la red, una asociación de SCTP con un número de trayectorias puede comportarse semejantemente como el mismo numero de las conexiones en TCP.

Como en el TCP, SCTP tiene dos modos, start slow y  evitar congestiones. El modo es determinado por un sistema de variables del control de congestión, y como se ha mencionado ya, éstos son específicos de la trayectoria. Así pues, mientras que la transmisión a la trayectoria primaria puede estar en el modo de evitar congestiones, SCTP puede utilizar start slow para la(s) trayectoria(s) de reserva.

Cuando los datos son entregados y reconocidos, el CWND se aumenta

[31] Congestion Window

constantemente, y una vez que exceda un cierto límite llamado SSTRESH[32], se realiza un cambio de modo, de start show a evitar congestiones. Generalmente, en el modo start slow, el CWND se aumenta más rápidamente (casi un MTU por cada SACK chunk), y de modo de la evitar congestiones, es aumentado solamente en un MTU por cada RTT[33].

Los eventos que causan retransmisiones (timeouts o retransmisiones rápidas) hacen que el SSTHRESH se reduzca drásticamente, y reinician el CWND (donde un timeout causa un nuevo start slow con CWND=MTU, y una retransmisión rápida coloca el CWND=SSTHRESH).

### 2.2.4 MULTIHOMING

Una característica esencial de SCTP es el soporte para nodos multihomed, es decir los nodos que se pueden alcanzar bajo varias direcciones del IP. Si se configuran los nodos de SCTP y la red del IP de una manera tal que el tráfico a partir de un nodo a otro pueda realizar su viaje físicamente a través de trayectorias diferentes, las asociaciones llegan a ser tolerantes contra fallas físicas de la red y otros problemas de ese tipo.

La figura 7 ilustra este tipo de conexiones, un nodo con varios proveedores (uno principal y otros de redundancia) que tiene la posibilidad de mantener su conexión a la red incluso si alguno de sus proveedores falla en la prestación del servicio, gracias a que los demás proveedores aun le ofrecen trayectorias de conexión.

[32] Slow Start Threshold
[33] Round Trip Time (Tiempo de ida y vuelta de los paquetes)

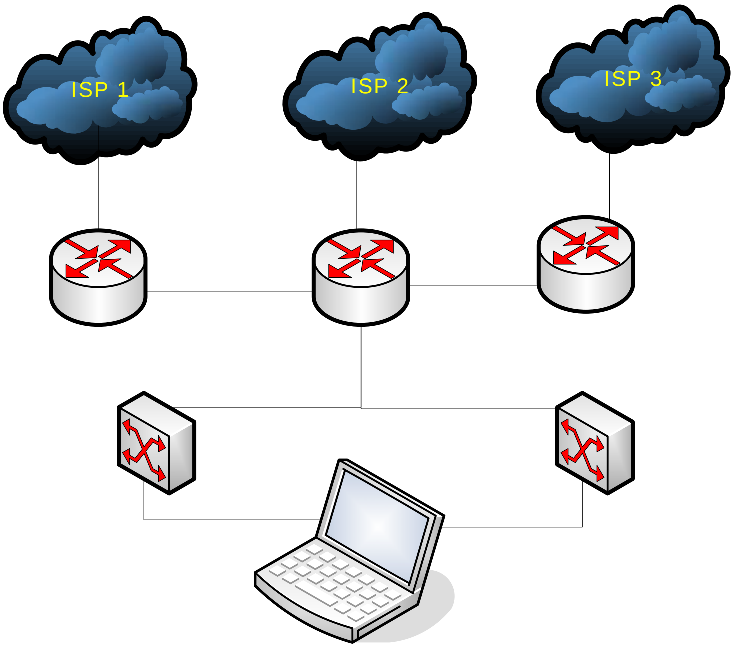


**Figura 7 Ejemplo de sistemas Multihomed**

Si un cliente es multihomed, informa al servidor sobre todas sus direcciones del IP con los parámetros de dirección del *chunk* INIT. Gracias a esto, se requiere al cliente solamente saber la dirección IP del servidor porque el servidor proporciona todas sus direcciones del IP al cliente en el *chunk* INIT-ACK. SCTP puede manejar las direcciones de la versión 4 y 6 de IP (incluso mezcladas). SCTP mira cada dirección IP de su par como una "trayectoria de transmisión" hacia un punto final.

Si no se contiene direcciones explicitas en el *chunk* INIT o de INIT-ACK, la dirección IP de la fuente del paquete del IP que lleva el datagrama de SCTP es utilizada. Esto facilita el uso de SCTP cuando se utiliza NAT[34]. Para facilitar esto aun mas, se ha introducido una característica opcional adicional en el RFC2960 la cual permite el uso de los nombres de host o en vez de direcciones del IP.

Una instancia de SCTP supervisa todas las trayectorias de transmisión al par de una asociación. Con este fin, los *chunks* HEARTBEAT se envían sobre todas las trayectorias que no se estén utilizando actualmente para la

[34] Network Address Translation

transmisión de los chunks de datos. Cada *chunk* HEARTBEAT tiene que ser reconocido por un *chunk* HEARTBEAT-ACK.

A cada trayectoria se le asigna un estado: es activa o inactiva. Una trayectoria es *activa* si se ha utilizado recientemente para transmitir un datagrama (arbitrario) de SCTP que ha sido reconocido por el par. Si las transmisiones en cierta trayectoria se parecen fallar en varias ocasiones, la trayectoria se mira como *inactiva*.

El numero de eventos donde los heartbeats no fueron reconocidos dentro de cierto tiempo, o  de la retransmisiones se cuenta con base en la asociación, y si se excede cierto limite (el valor de el cual puede ser configurable), el punto final del par se considera inalcanzable, y la asociación será terminada.

*Selección de trayectoria*

En la creación de una asociación SCTP, una de las direcciones del IP de la lista vuelta se selecciona como *trayectoria primaria* inicial. Los *chunks* de datos son transmitidos sobre esta trayectoria de transmisión primaria por defecto. Para las retransmisiones sin embargo, otra trayectoria activa puede ser seleccionada, si alguna esta disponible. Para apoyar la medición del retardo de viaje de los paquetes, los *chunks* SACK se deben enviar a la dirección de la fuente del paquete del IP que lleva el *chunk* de datos que accionó el SACK.

Los usuarios de SCTP están informados sobre el estado (estado y las medidas) de una trayectoria de transmisión a petición o cuando una trayectoria de transmisión cambia su estado.

### 2.2.5 CANALES

Mientras que TCP junta la transferencia confiable de los datos del usuario y la entrega estricta de dichos datos en un orden especifico, SCTP separa la transferencia confiable de datagramas del mecanismo de entrega. Esto permite adaptar uso del protocolo a las necesidades específicas de las implementaciones que usen SCTP. Algunos usos pueden necesitar solamente ordenar parcialmente la entrega de datagramas mientras que otros se pueden incluso satisfacer con una transferencia confiable que no garantiza ningún mantenimiento de la secuencia de entrega.

SCTP distingue diversos *canales* de mensajes dentro de una asociación de SCTP. Esto permite un esquema de la entrega donde solamente la secuencia de mensajes necesita ser mantenida por el canal (entrega en secuencia parcial) que reduce la cabecera de separación entre los canales independientes de mensajes. Además, SCTP proporciona un mecanismo para puentear el servicio de entrega ordenado, para entregar mensajes al usuario de SCTP tan pronto como se reciban totalmente (entrega según orden de llegada).

El control de flujo y de la congestión en SCTP se han diseñado de una manera que asegura ese trafico de SCTP se comporta en la red de la misma manera que lo hace el trafico del TCP. Esto permite una introducción de los servicios de SCTP en redes existentes del IP

### *2.3. MGCP / H.248 (MEGACO)*

MGCP es un protocolo tipo maestro-esclavo para comunicaciones entre elementos de control de llamada y gateways de telefonía IP. MGCP surgió con el objeto de facilitar la integración del protocolo de SS7 con VoIP, ya que la arquitectura definida en H.323 es incompatible con el mundo de los servicios de telefonía publica. Las soluciones de VoIP basadas en MGCP separan la inteligencia de la llamada del manejo de los medios, lo cual lo hace bastante apropiado para la tecnología softswitch.

Megaco (o su equivalente la recomendación H.248 de la UIT) es bastante similar a MGCP desde el punto de vista de la arquitectura y la relación controlador-gateway, pero también soporta  otras redes como ATM. Propuesto conjuntamente por el Grupo 16 de UIT y el IETF, Megaco añade a MGCP capacidades de interoperabilidad entre iguales, y proporciona un medio de control apropiado para dispositivos telefónicos IP que operen como maestro/esclavo.

Megaco explota el modelo gatekeeper y desplaza el control de señalización del gateway, hacia un "gateway de control de medios " o "softswitch". MGCP/Megaco es el protocolo usado para comunicaciones entre el controlador (MGC ) y el gateway de medios (MG), y está diseñado para el control remoto intradominio de dispositivos orientados a conexión o a sesión, tales como gateways VoIP, servidores de acceso, multiplexores de acceso DSL (DSLAMs), dispositivos enrutadores, MPLS, etc.

Esta basado en el principio de que toda la inteligencia de procesamiento reside en el MGC. El MG no retiene información del estado de la llamada, solo provee la capacidad de interconectar varias clases de medios bajo el control del MGC y entonces detecta y transmite varias clases de señalizaciones asociadas con esos medios. Megaco ve el MG como una colección de terminaciones, cada una de las cuales representa cierto tipo de medio.

Una terminación puede ser una entidad física estática como una línea analógica o una señal digital; o también puede ser una entidad lógica como un canal de paquetes de VoIP. Las terminaciones lógicas pueden ser creadas y destruidas mediante comandos de Megaco. Las interconexiones dentro del MG son creadas por medio de comandos de Megaco que exigen que dos o más terminaciones sean colocadas en el mismo contexto. Si los canales de medios asociados con terminaciones que están en el mismo contexto son de tipos diferentes (por ejemplo, uno es un *time slot* de un E1 y mientras que el otro es un canal de paquetes VoIP) entonces se espera que el MG realice las conversiones apropiadas entre ellos.

Para dar soporte a esto, las terminaciones tienen varias propiedades en sus canales de medios, asociadas con ellos como la identidad de la codificación de voz que ha de ser utilizada. Las terminaciones tienen otras propiedades, como una lista de eventos de señalización que deben ser notificados al MGC, y una lista de señales que son capaces de transmitir cuando haya una petición del MGC. Por ejemplo, una terminación de línea análoga debe ser capaz de notificar al MGC cuando están sucediendo eventos de descuelgue y de cuelgue; también debe ser capaz de aplicar timbres en la línea cuando el MGC lo indique.

Mediante el protocolo Megaco, el MG, al detectar un descuelgue (cuando una persona levanta el teléfono para hacer una llamada), se lo comunica al MGC. Este puede responder con un comando de instrucción al MG para que envíe tono de marcación y 'escuche' los tonos del número marcado. Después de detectar el número, el MGC determina como encaminar la llamada y, usando un protocolo de señalización inter-MGC como H.323 o SIP, contacta con el MGC del  terminal distante.

Los eventos y señales que están asociados con un tipo específico de terminación son descritos en un paquete. Megaco esta diseñado para ser un protocolo extensible, e incluye un mecanismo para permitir la especificación y registro de nuevos paquetes. Esta extensibilidad supera una gran desventaja de muchos protocolos de control de medios más antiguos como MGCP, ya que direcciona la necesidades de los protocolo de paquetes de voz diferentes a VoIP y proporciona los medios para introducir variaciones para los servicios de telefonía analógica en cada país.

En general existen dos filosofías de trabajo en lo referente al funcionamiento de una red pública de VoIP. SIP es el protocolo adecuado para los que piensan que los terminales deben tener suficiente inteligencia para que incluyan la señalización, pues el control de la llamada se ejecuta directamente en el terminal. Para las grandes redes, o los que siguen la filosofía de separar la señalización de control de llamada en un computador central, se requiere un protocolo entre el MG y la parte de señalización (MGC), y aquí es más apropiado MGCP / Megaco.

# **Capítulo 3**
# Protocolos de Transmisión de Datos

*Obtenere nociones básicas de los diferentes protocolos utilizados para realizar la transmisión de los paquetes de voz, video y datos en redes IP.*

## 3 PROTOCOLOS DE TRANSMISIÓN DE DATOS.

La importancia de los protocolos de transmisión dentro del funcionamiento del un softswitch que soporte transmisiones de voz, datos y video es sumamente alta y por tanto se dará una breve explicación del funcionamiento de los protocolos mas importantes.

### *3.1 MODELO DE ARQUITECTURA H.323.*

El estándar H.323 especifica los componentes, protocolos y procedimientos para las comunicaciones multimedios sobre redes de paquetes. H.323 es una recomendación general de la UIT que especifica las normas para comunicaciones sobre redes de área local (LAN), sin proporcionar calidad de servicio. Forma parte de la serie H.320 de recomendaciones sobre videoconferencia.

Aunque fue concebido inicialmente para videoconferencia entre computadores, las normas H.323 sirven de base a muchos productos de telefonía IP y videotelefonía. El esquema siguiente muestra la relación entre protocolos H.323.

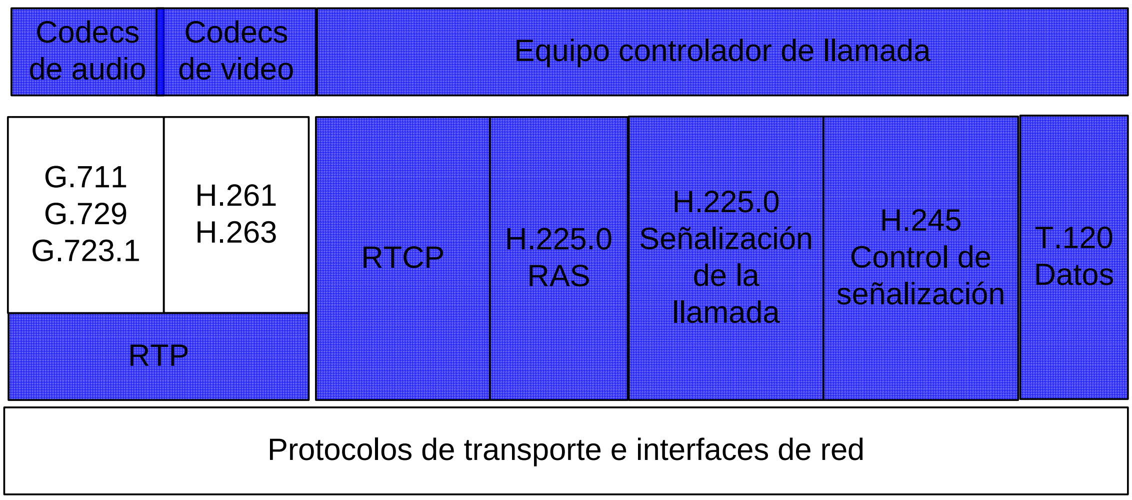


**Figura 8 Protocolos H.323 (en azul)**

En la arquitectura H.323 se definen cuatro componentes funcionales que cuando se colocan juntos en red, proveen servicios de comunicación multimedia punto a punto y punto a multipunto, Estos componentes son:

1. Terminal
2. Gateway
3. Gatekeeper (o Gateway de Control)
4. Unidades de Control Multipunto (MCU[35]).

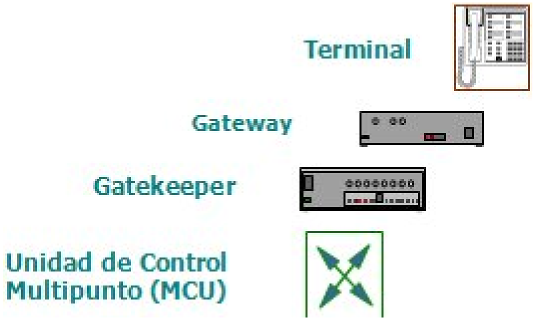


**Figura 9 Componentes de red H.323**

### 3.1.1 TERMINAL

Es el extremo de cliente que proporciona comunicación bi-direccional. Todos los terminales H.323 soportan comunicaciones de voz y, opcionalmente, vídeo y datos. H.323 especifica los modos de operación requeridos para el inter-funcionamiento de diferentes terminales de audio, video y/o datos. Los terminales utilizan el protocolo H.245 para la negociación de canales y capacidades, así como Q.931 (H.225) para señalización de llamadas y comunicación con el controlador de llamadas (Gatekeeper) por medio del protocolo RAS[36]. Por último para la transmisión de audio/vídeo paquetes los terminales deben soportar RTP/RTCP.

Los terminales H.323 deben soportar:

- H.245 para el intercambio de las capacidades de las terminales y la creación de canales de medios.

[35] Multipoint Control Units
[36] *Registration / Admission / Status*

- H.225 para la señalización y establecimiento de llamadas.
- RAS para el registro y otros controles de admisión con un gatekeeper.
- RTP/RTCP para los paquetes de audio y video secuenciados.

Los terminales también deben soportar el CODECs de audio G.711. Otros componentes opcionales en un terminal son los CODECs de video, T.120 (protocolo de conferencia de datos) y capacidades de MCU.

### 3.1.2 GATEWAY

Es un elemento opcional que sirve de traductor de funciones entre un extremo de conferencia y otro terminal. Refleja las características de un punto de terminación de red de área local en uno de red conmutada y viceversa. Contiene las funciones de traducción de protocolos y transcodificación de audio y vídeo entre el lado RTCP y lado IP. El procesamiento que realiza el gateway de la cadena de audio que atraviesa una red IP es transparente para los usuarios.

Adicionalmente, el gateway realiza las funciones de establecimiento y terminación de llamadas, tanto desde el lado de red LAN como desde el de conmutación de circuitos. Así, la persona que realiza una llamada desde un teléfono convencional ingresa a un gateway marcando el número de acceso a un sistema de voz interactiva, o bien el numero de destino, según se haya establecido el proceso de autenticación, y una vez admitido prosigue el proceso y escucha los tonos de llamada habituales.

Los terminales se comunican con los gateways utilizando el protocolo de control de señalización H.245, el gateway traduce este protocolo a una forma transparente para su contraparte no-H.323 y viceversa. El gateway también realiza el establecimiento y la liberación de los canales tanto en la red H.323 como en la parte no-H.323 de la red. La traducción entre los formatos de datos de audio, video y datos también se puede realizar en el gateway, sin embargo, las traducciones de audio y video pueden no ser requeridas si ambos terminales tienen modos de transmisión comunes. Por ejemplo, en el caso de un gateway entre terminales H.320 e ISDN, ambas terminales requieres G.711 para audio y H.261 para video, por tanto existe un modo común y no es necesaria la traducción.

Los gatekeepers saben cuales puntos finales son gateways debido a que esto se les indica cuando los terminales y los gateways se registran con el gatekeeper. Un gateway puede soportar múltiples llamadas ente redes H.323 y no-H.323. Por ultimo, un gateway es un componente lógico y puede ser implementado como una parte del gatekeeper o de un MCU.

### 3.1.3 GATEKEEPER

Proporciona servicios de control de llamada, actuando en cierta forma como un conmutador virtual, dentro de sus funciones esta la traducción de direcciones y el manejo del ancho de banda como se define en RAS. Estos elementos son opcionales en las redes H.323, sin embargo, si están presentes en la red los terminales y los gateways deben usar sus servicios. El estándar H.323 define los servicios mandatarios que el gatekeeper debe proveer y especifica otras funcionalidades opcionales que pueden tener.

Una característica opcional que pueden tener es el enrutamiento de las llamadas. Los puntos extremos de la red envían mensajes de señalización al gatekeeper, el cual los re-direcciona hacia su destino, de forma alternativa los extremos pueden enviar mensajes directamente a sus extremos pares. Esta característica de los gatekeepers es muy valiosa, ya que el monitoreo de las llamadas realizado por este provee un mejor control de las mismas. Por ultimo se debe aclarar que el gatekeeper es también un elemento lógico, y como tal puede ser una aplicación en una computadora o puede estar integrado en un gateway o un terminal H.323, y realiza las funciones de:

- Enrutamiento mediante traducción de direcciones (nombre-dirección IP).
- Control de admisión, mediante mensajes de petición y confirmación o rechazo.
- Control de petición de ancho de banda en la comunicación.

- Gestión de zona: Gestiona los gateways, terminales y unidades de control multipunto registradas en la zona de control del gatekeeper.

Adicionalmente un gatekeeper puede tener las funciones de señalización de llamadas (Q.931), autorización y gestión de llamadas, gestión del ancho de banda disponible, servicio de directorios, etc. Lo gateways conectan con los gatekeepers de VoIP mediante enlaces estándar H.323v2, utilizando el protocolo RAS (H.225). De esta manera actúan como controladores del sistema y cumplen con el segundo nivel de funciones esenciales de un sistema de VoIP de clase carrier, a saber:

- Autenticación mediante control de admisión
- Enrutamiento del servidor de directorios
- Contabilidad de llamadas
- Determinación de tarifas.

Los gatekeepers acceden al servidor *backend* del centro de cómputos del operador para autenticar a los que llaman como abonados válidos al servicio, optimizar la selección del gateway de destino y sus alternativas, y hacer un seguimiento y actualización de los registros de llamadas, incluyendo los detalles del plan de facturación de la persona que efectúa la llamada.

### 3.1.4 UNIDAD DE CONTROL MULTIPUNTO

Es el elemento que controla las conferencias entre tres o más terminales, ya sea de manera centralizada o distribuida. Se compone de un Controlador Multipunto y uno o más Procesadores Multipunto; el primero determina y maneja los recursos de la multiconferencia mediante las funciones de control H.245. Opcionalmente, el Procesador Multipunto conmuta, mezcla y procesa los flujos de vídeo y audio con bits de datos. La función de un MCU puede estar integrada en un terminal, gateway o gatekeeper.

H.323 proporciona interoperabilidad entre dispositivos, aplicaciones y fabricantes, lo que permite que los productos que cumplan con H.323 puedan operar entre sí. La recomendación H.323 incluye también normas para la compresión y descompresión de datos audio y vídeo, de tal forma que los equipos de diferentes fabricantes tengan un área de soporte común.

H.323 comprende una serie de estándares y se apoya en protocolos que cubren los distintos aspectos de la comunicación VoIP, indicados a continuación.

- **H.225 RAS**

Es el protocolo entre puntos extremos (terminales y gateways) y gatekeepers. Permite a una estación H.323 localizar otra estación H.323 a través del gatekeeper. El RAS es usado para llevar a cabo el registro, el control de admisión y cambios de ancho de banda y estado. Un canal RAS es usado para intercambiar mensajes RAS, este canal de señalización es abierto entre un punto extremo y un gatekeeper antes del establecimiento de cualquier otro canal.

- **H.225 Señalización de llamadas.**

La señalización de llamadas de H.225 es usada para establecer una conexión entre dos puntos extremos H.323. Esto se consigue intercambiando mensajes del protocolo H.225 en el canal de señalización de llamada. Este canal es abierto entre dos puntos extremos H.323 o entre un punto extremo y un gatekeeper. También existe el protocolo Q.931 que contiene los mensajes de señalización inicial y terminación de llamada.

- **H.245 Control de señalización**

Este protocolo es usado para intercambiar mensajes de control de extremo a extremo para gobernar la operación del punto extremo H.323. Estos mensajes de control contienen información relacionada con:

- Capacidades de intercambio.
- Abrir y cerrar los canales lógicos utilizados para llevar las ráfagas de medios.
- Mensajes de control de flujo.
- Comandos generales e indicadores.

- **Compresión de Voz:**

1. Codecs requeridos: G.711 y G.723
2. Codecs opcionales: G.728, G.729 y G.722

| Recomendación | G711 | G722 | G728 | G729 | G723.1 |
|---|---|---|---|---|---|
| Tipo de código | MIC compandido | ADPCM | LD-CELP | CS-ACELP | MPC, MCQ y ACELP |
| Velocidad | 64kbps | 32-64 kbps | 16kbps | 8kbps | 6.3 y 5.3 kbps |
| Complejidad (MIPS) | <<1 | ~1 | 5 | 20 | 18 |
| RAM | 1 byte | <50 bytes | 1Kb | 2Kb | 2.2Kb |

**Cuadro I**: Codecs de voz utilizados en H.323

Las comunicaciones H.323 pueden ser una combinación de audio, vídeo, datos y señales de control. Las capacidades de audio, señalización Q.931 para establecimiento de llamada, RAS para control de acceso, y el canal de control H.245 son requerimientos esenciales, mientras que otras capacidades como conferencia de vídeo y datos son opcionales. Los terminales H.323 pueden operar de manera asimétrica (algoritmos de codificación y decodificación diferentes) y pueden enviar/recibir en mas de un canal.

## 3.2 PROTOCOLOS PARA TRANSMISIÓN DE VOZ SOBRE REDES IP

El servicio básico que debe prestar un softswitch, como se ha mencionado antes, debe ser el servicio de voz, por esta razón a continuación se da una breve explicación de los protocolos más importantes y más utilizados para el transporte de voz sobre redes IP.

La solución para enviar tráfico de voz (y también de vídeo) sobre Internet consiste básicamente en asignar prioridades por tipo de tráfico y hacer reservas de ancho de banda. El modelo propuesto por el IETF para tiempo real en redes IP permite a las aplicaciones multimedios compartir la infraestructura de aplicaciones convencionales, teniendo en cuenta el uso de protocolos como **RTP/RTCP y RSVP**.

### 3.2.1 RTP (*Real Time Transport Protocol*)

Este protocolo está diseñado para el transporte de datos en tiempo real sobre redes de datagramas, por lo que es el medio apropiado para transportar datos de audio y vídeo. Se puede utilizar tanto para la transferencia de datos en un solo sentido, como en la recepción de vídeo (*streaming*) como en servicios interactivos (telefonía IP), sea en modo punto a punto o multipunto (multicast).

Para efectuar el transporte de datos en tiempo real el paquete RTP tiene una cabecera de 20 bytes, extensible, con funciones para la reconstrucción de secuencias, detección de fallos, seguridad e identificación de contenido. Entre los campos de la cabecera se incluyen las marcas de tiempo (*timestamp*) y los números de secuencia**,** que sirven para sincronizar los datagramas UDP recibidos fuera de secuencia o fragmentados y reconstruirlos a la velocidad adecuada. Otra función que realiza RTP es la identificación del tipo de contenido, que detecta el formato de los datos para ajustarse a la disponibilidad de ancho de banda, identificando por ej., si es PCM, vídeo/audio MPEG o flujos de vídeo H.261. Otro campo significativo es el que identifica el origen o fuente de sincronismo, que constituye la base de tiempos común a todos los participantes.

El paquete RTP, incluyendo su cabecera, se transmite normalmente encapsulado en datagramas IP como muestra la figura siguiente, aunque también se puede usar con otros protocolos como ATM.

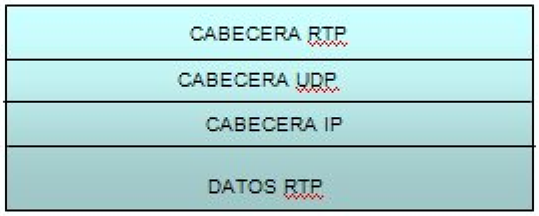


**Figura 10 Formato de Paquete RTP**

El protocolo auxiliar RTCP sirve, junto con RTP, para controlar la calidad de la transmisión de datos y participantes en la sesión. Con la información de QoS que proporcionan los receptores mediante RTCP, el origen de llamada puede por ejemplo ajustar su velocidad, mientras otros receptores pueden determinar si los problemas de QoS son locales o de la red. En RTCP se pueden definir cuatro paquetes principales de control:

1. SR (*Sender Report)*, informe que agrupa las estadísticas de transmisión (pérdidas de paquetes en la sesión, variación del retardo, etc.).
2. RR (*Receiver Report*), conjunto de estadísticas sobre la comunicación entre participantes, emitidos por los receptores de una sesión.
3. SDES (*Source Description*), que describe la fuente de la llamada.
4. BYE, mensaje de fin de participación en una sesión.

### 3.2.2 RSVP (*Resource Reservation Protocol*)

RSVP es un protocolo de control de red que aporta calidad de servicio (QoS) extremo a extremo a un flujo de datos IP. Las aplicaciones de comunicaciones en tiempo real reservan los recursos necesarios de encaminamiento, de tal manera que durante la transmisión siempre este disponible el ancho de banda necesario. Es una solución distribuida, que permite a múltiples receptores heterogéneos efectuar reservas específicamente dimensionadas a sus propias necesidades.

Cuando el receptor de datos requiere una calidad de servicio específica, utiliza RSVP para solicitar reserva de recursos a los enrutadores a lo largo del trayecto de los datos. El protocolo negocia los parámetros de conexión con los enrutadores y mantiene los estados de éstos y de los hosts. El modo de atribución de recursos tiene la ventaja de que, al ser efectuado por el receptor, puede demandar una QoS adaptada a sus necesidades y al consumo deseado, por lo que se puede 'asegurar' una QoS y acordar un nivel de servicio.

RSVP no es un protocolo de enrutamiento, pues en el proceso de reserva no transmite los datos simultáneamente, y exige que los sistemas terminales funcionen en modo conectado. Sin embargo, para garantizar un ancho de banda determinado debe conocer previamente a donde dirigir las peticiones de reserva de recursos. Los flujos se transmiten en modo simplex, es decir, que solamente reserva recursos para transmisión en un sentido (hacia el receptor que solicita la reserva).

En el gráfico adjunto se resumen los procedimientos que utiliza el protocolo para reserva y estado de los recursos.

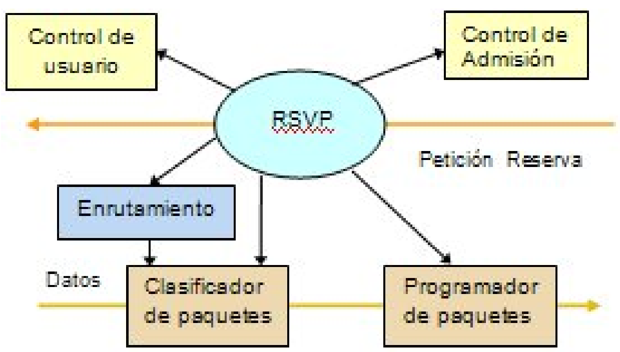


**Figura 11 RSVP. Proceso de petición y reserva de recursos**

En la figura se aprecian dos caminos: uno de petición de reserva y otro en sentido contrario de envío de mensajes que indican el trayecto a seguir por los datos para que el receptor (o los receptores) pueda(n) determinar los recursos que se deben reservar, según el trayecto seguido desde el origen de los datos.

El control de usuarios incluye los permisos de acceso y autenticación. El control de admisión se refiere al control de los recursos necesarios para suministrar la calidad de servicio (QoS) solicitada.

RSVP comprueba el estado de los procedimientos de control de usuarios y admisión, y si son válidos manda los parámetros correspondientes al clasificador de paquetes (para ordenarlos según prioridad) y al programador de paquetes (para transmitirlos una vez clasificados). También se comunica con el proceso de enrutamiento para determinar el trayecto a enviar sus peticiones de reserva y manejo de usuarios y rutas.

Al ser RSVP un protocolo diseñado para multicast, los receptores que inician la petición de reserva pueden cambiar dentro de un grupo, y también las rutas reservadas pueden ser modificadas sin gran sobrecarga para el emisor. En definitiva, RSVP está orientado a recepción, también es compatible con IPv6 y se adapta sin dificultad a diferentes calidades de servicio requeridas para un mismo emisor. Por otro lado, aunque el principio de reserva de recursos es innovador en el mundo IP, su utilización requiere el empleo de muchos recursos y de arquitecturas probablemente demasiado complejas para ser administradas en redes de área extensa.

### ***3.3. SIP (IETF RFC 2543)***

Aunque H.323 es la referencia para interoperabilidad en el mundo de VoIP, existen otros protocolos alternativos de menor complejidad que también se utilizan en aplicaciones de telefonía sobre IP. El protocolo SIP (*Session Initiation Protocol*) es uno de ellos, que por ser relativamente simple, claro y familiar en el ámbito de las aplicaciones Internet está siendo utilizado con frecuencia creciente.

SIP es un protocolo de señalización de llamadas basado en texto ASCII. Está diseñado teniendo en cuenta protocolos textuales establecidos por el IETF en Internet, tales como SMTP y HTTP, con codificación normalizada flexible y extensible. Comparte otros elementos comunes a Internet, como los nombres DNS y direcciones de correo electrónico. Utiliza, como en HTTP, el modelo "petición-respuesta" en la iniciación de una llamada, que puede ser establecida estrictamente sin la mediación de un agente de llamada. Algunas de las características más importantes de SIP son:

- Determina la localización del punto de destino (enruta) ya sea mediante la resolución de direcciones, el mapeo de nombres o el redireccionamiento de llamadas., de forma manual o automática, permitiendo la movilidad personal.
- Determina las capacidades de transmisión de medios hasta el destino (mediante SDP[37])
- Determina la disponibilidad del punto de destino (SDP). Si una llamada no puede ser completada debido a que el destino es inaccesible, SIP determina si este ya esta utilizando la línea o no respondió luego de un cierto número de timbres, para luego enviar un mensaje indicando porque el destino fue inaccesible.
- Determina el "nivel mínimo" de servicios comunes entre dos puntos extremos. Las conferencias son establecidas solo con especificaciones que pueden ser soportadas por todos los puntos extremos.
- SIP puede establecer, modificar y terminar llamadas y sesiones multimedia. Maneja la terminación y transferencia de llamadas. También soporta cambios en el intermedio de una sesión como la adición de otro extremo o el cambio de características del medio o incluso un cambio de codecs.

[37] *Session Description Protocol*

- Soporta servicio de redireccionamiento de nombres de manera transparente.
- Proporciona control de la llamada (retención, transferencia, cambio de medio...)
- Puede interactuar con otros servicios de aplicaciones como LDAP[38], XML, servidores de localización o aplicaciones de bases de datos. Estas aplicaciones proveen servicios finales como directorio, autenticación y facturación.
- Puede manejar sesiones multicast, (conferencia a tres o más).

SIP es independiente del nivel de paquetes. El protocolo es abierto y escalable, designado como un protocolo de propósito general. Por ello se necesitan extensiones a SIP para que el protocolo sea verdaderamente funcional e ínter operable.

Las entidades principales en SIP son el Agente de Usuario (AU), o terminal (puede ser un teléfono IP, un gateway o un programa de telefonía), y el Servidor SIP.

Una sesión básicamente se origina cuando se quieren comunicar dos agentes de usuario (clientes). El AU que llama, lanza una petición en forma de texto simple, que el llamado debe responder aceptando o rechazando la invitación. El usuario llamante envía entonces un reconocimiento de vuelta que indica está preparado para enviar y recibir los datos audio/video.

Los servidores SIP proporcionan puntos de acceso singulares para localizar a los usuarios, traducir nombres y direcciones, encaminar los mensajes de señalización entre agentes de usuario y redirigir las peticiones. Existen dos tipos posibles de servidores SIP:

- **Servidor Proxy**. Es el único punto de contacto del AU para los mensajes de señalización, recibe peticiones de los clientes y los dirige a otros servidores

[38] Lightweight Directory Access Protocol

o al cliente destino. Puede ramificar una petición de llamada hacia varias direcciones simultáneamente.

- **Servidor de Redireccionamiento.** Acepta peticiones SIP y da a conocer la dirección del AU llamado, no interviniendo más en la sesión.

- **Servidor Registrar.** Procesa los pedidos de las AU para registrar su localización actual. Los servidores registrar son normalmente colocados junto (en la misma maquina) a los servidores de redireccionamiento o de Proxy.

### 3.3.1 FUNCIONAMIENTO DE SIP

SIP es un protocolo simple basado en código ASCII, que usa peticiones y respuestas para establecer una comunicación entre varios componentes de la red y establecer una conferencia entre dos o más puntos extremos. Los usuarios de una red SIP se identifican con una dirección SIP única, la cual es similar a una dirección de *e-mail* ya que tiene el formato: *userID@gateway.com*. El *userID* puede ser un nombre o una dirección E.164.

El usuario se registra con un servidor registrar usando su dirección SIP asignada. El servidor registrar proporciona esta información al servidor de localización a pedido, cuando un usuario inicia una llamada, una petición SIP es enviada a un servidor SIP (ya sea un servidor Proxy o de redireccionamiento). La petición incluye la dirección de quien origina la llamada, y la dirección de quien debe recibirla.

Con el tiempo, un usuario final SIP se puede mover entre sistemas terminales, la localización del usuario final puede ser registrada dinámicamente con el servidor SIP. El servidor de localización puede usar uno o más protocolos (incluyendo rwhois y LDAP) para localizar al usuario final, debido a que el usuario final puede iniciar sesión en mas de una estación y debido a que el servidor de localización puede tener información imprecisa, puede regresar mas de una dirección para el usuario final. Si la petición viene a través de un servidor Proxy, este intentara con cada una de las direcciones recibidas hasta que encuentre al usuario final. Si la petición viene a través de un servidor de redireccionamiento, este envía todas las direcciones a quien inicio la llamada en el campo “Contactos” de la cabecera de la respuesta de la invitación.

### 3.3.2 PROTOCOLOS SIP

SIP sirve para las funciones básicas de establecimiento y terminación de llamada. La transferencia de datos se realiza utilizando los protocolos de transporte RTP/RTCP y la configuración de una sesión se describe siguiendo **SDP**.

SDP está pensado para describir sesiones multimedia como el anuncio, invitación u otras formas de inicio de una sesión. Las descripciones pueden incluir el nombre de la sesión y su propósito, duración, información de recursos como medios de transmisión, información para la recepción y, en general, toda aquella necesaria para poder unirse a la sesión.

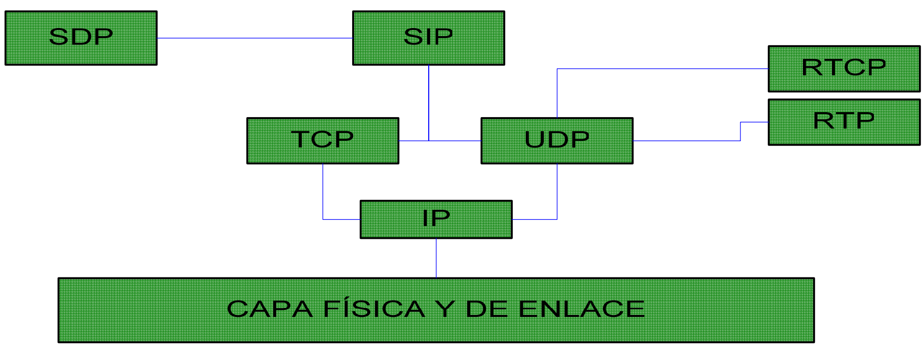


**Figura 12 Protocolos SIP**

SIP puede ser transportado sobre cualquier protocolo de menor nivel, como UDP, TCP, ATM y Frame Relay. Sin embargo, normalmente se sirve sobre TCP/IP debido a que ya existe una amplia conectividad, servicios de directorio, de nombres y una base bien conocida. En la fig. 15 se muestra SIP sobre TCP/IP, y en la tabla I se compara con H.323.

# Capítulo 4

# Escenarios de Aplicación de Softswitch.

*Mostrar diferentes configuraciones de comunicación de elementos de la red utilizando las propiedades de la tecnología Softswitch para transmisión de datos en una red IP.*

## 4. ESCENARIOS DE APLICACIONES DEL SOFTSWITCH

Los protocolos mencionados, son utilizados por lo general en función de las redes y aplicaciones de telefonía que lo soportan. Así, H.323 es más apropiado para llamadas por Internet, ya que esta no ofrece calidad de servicio. Para inter-funcionamiento con la RTPC o en comunicaciones IP con calidad de servicio se tiende a utilizar H.248/Megaco o SIP, en función del grado de distribución del control de las llamadas y las funcionalidades requeridas al sistema.

### *4.1 TRONCAL IP PARA LLAMADAS CONVENCIONALES*

En las comunicaciones a larga distancia principalmente, con objeto de abaratar costos y mejorar eficiencia de uso de ancho de banda, cada vez se utiliza más la comunicación entre teléfonos convencionales utilizando la tecnología IP en la red de tránsito. Es el denominado *trunking* (troncal) IP. En este caso es necesario la función de interconexión (gateway) para unir las redes de acceso con la de tránsito IP (ver Fig. 16). La tecnología empleada en la llamada es totalmente transparente al usuario, la red IP debe propagar la señalización SS7, con sus capacidades y servicios básicos, esta es una configuración que probablemente se ha estado utilizando en las llamadas internacionales por muchos operadores.

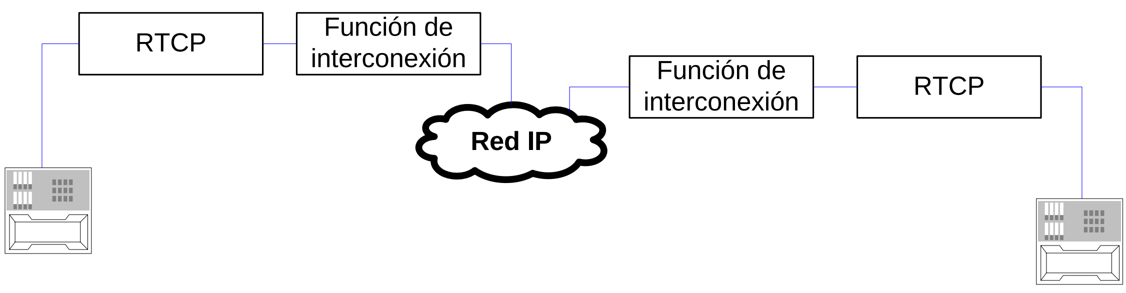


**Figura 13 Comunicación Teléfono a Teléfono con transito IP**

El desarrollo de llamadas de teléfono a teléfono utilizando la tecnología softswitch se puede sintetizar en la siguiente representación, donde el usuario

A tiene un número de abonado 6671660 y el usuario B tiene un número de abonado 5893652.

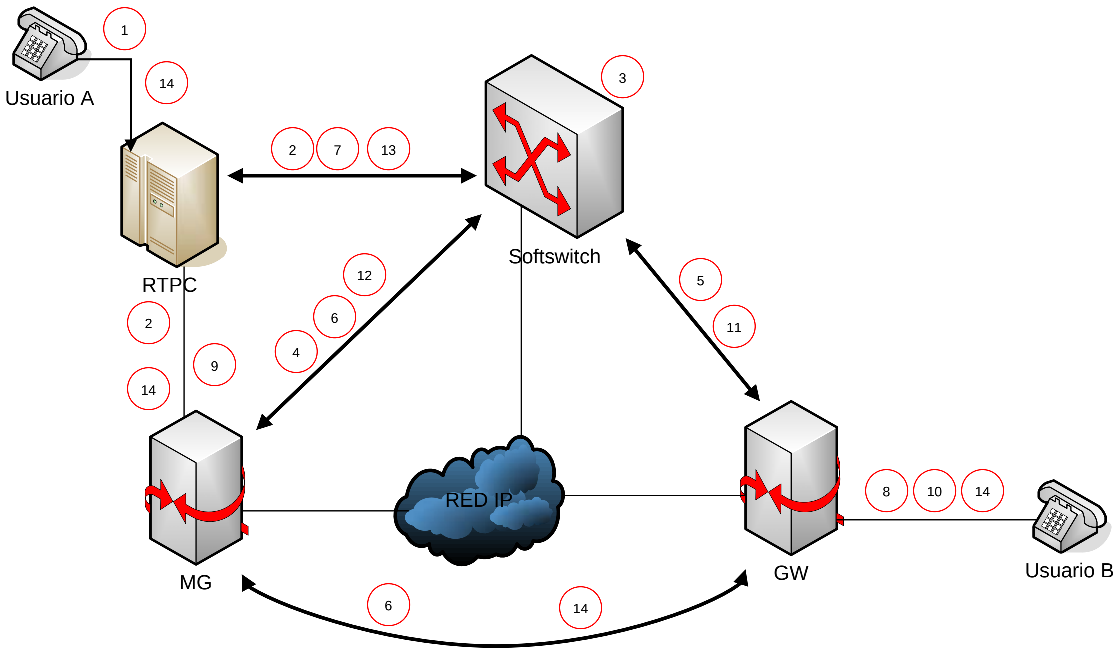


**Figura 14 Comunicación controlada por Softswitch**

1. El usuario A intenta iniciar una comunicación hacia un usuario B con número de abonado 5893652.

2. La central telefónica que recibe la señal del usuario A, analiza la numeración recibida. Según su tabla de enrutamiento todos los números telefónicos de la serie 58XXXXX deben encaminarse a través del MG y por tanto toma un circuito en dicha ruta. Las comunicaciones por dicha ruta debe señalizar con el softswitch, entonces envía un mensaje IAM[39] de señalización SS7 al softswitch para iniciar la comunicación.

3. El softswitch recibe el mensaje IAM, según el cual se requiere establecer una comunicación telefónica hacia el usuario identificado con el número 5893652 y la toma de un circuito con el MG cuyo CIC[40] va en el mensaje SS7. El softswitch analiza sus bases de datos y reconoce al usuario con

[39] Initial Address Message
[40] Código de Identificación de Circuito

el número 5893652 como propio e identifica el GW (Gateway de acceso) al que esta conectado.

4. El softswitch envía al MG, un comando de MEGACO, para que añada a una nueva instancia la siguiente información: una terminación RTP con la red IP y una terminación de circuito TDM con la central de la RTPC. Dentro del mismo comando, el softswitch indica los parámetros del medio: para la terminación TDM se indica el CIC y para la terminación RTP se indica el codec preferido a utilizar. El MG envía al softswitch, la respuesta al comando, indicando por su parte la dirección IP y el puerto UDP locales de la terminación RTP solicitada.

5. El softswitch envía al GW, un comando para que este añada una nueva instancia con la siguiente información: una terminación correspondiente a un puerto vocal analógico del MG y una terminación RTP con la red IP. Dentro del comando H.248, el softswitch indica los parámetros del medio: para la terminación del puerto vocal analógico del MG, se identifica el correspondiente al usuario con número de abonado 5893652 y para la terminación RTP, se indica la codificación a utilizar para la voz y la dirección IP y el puerto UDP del otro extremo. Además se indica para el puerto vocal analógico del usuario, que se le envíe el timbrado de llamada. Luego el GW envía al softswitch, la respuesta al comando, indicando por su parte la dirección IP y el puerto UDP locales de la terminación RTP solicitada.

6. El softswitch envía al MG, un comando para modificar la terminación RTP de la instancia existente para esta comunicación (conformado en el paso 4), mediante el agregado de la información de dirección IP y puerto UDP del otro extremo (información recibida por el softswitch en el paso anterior). También indica modificar la terminación del puerto TDM mediante el envío el tono de llamada hacia atrás. El MG envía al softswitch, la respuesta al comando acusando el recibo del mismo y a partir de este momento, el MG y el GW ya tienen una sesión de medio vocal a través de la red de transporte IP.

7. El softswitch envía un mensaje ACM[41] de señalización SS7 a la central telefónica del usuario A.

8. El GW realiza el timbrado de llamada sobre el puerto del usuario B y queda a la espera de que éste conteste.

9. El MG envía el tono de llamada hacia atrás por el puerto TDM para que el usuario A lo escuche y sepa que se está timbrando al usuario B.

10. El usuario B responde.

11. El GW envía al softswitch un comando donde se indica con respecto a la terminación correspondiente al puerto del usuario B, el evento de respuesta del usuario B, el softswitch envía al GW el acuse de recibo del comando junto con un comando indicando que termine el timbrado en la terminación correspondiente al puerto del usuario B. El GW envía al softswitch la respuesta a este último comando manifestando acuse de recibo del mismo.

12. El softswitch envía al MG un comando para indicar que se debe modificar la terminación del puerto TDM, mediante la suspensión del tono de llamada hacia atrás.

13. El softswitch envía un mensaje ANM[42] de señalización SS7 a la central telefónica de A, con lo cual dicha central sabrá que el usuario B respondió y pasará a tasar la llamada.

14. Se produce la conversación entre el usuario A y el usuario B a través del medio vocal.

[41] Address Complete Message
[42] Answer Message

A continuación se muestran algunas configuraciones típicas de llamadas que realizan transmisiones de voz sobre IP.

### 4.2 LLAMADAS PC a PC

En el caso de llamada entre computadores, las aplicaciones residentes se encargan de realizar la marcación por medio de un interfase de usuario adaptado (micrófonos y audifonos), así como la conversión de la voz en datos que circulan por la red IP (sea privada o pública). Para localizar a un usuario se requiere conocer su dirección IP y, naturalmente, que esté conectado a la red. Aquí el softswitch no necesita interactuar con una RTPC debido a que ambos puntos extremos pertenecen a una red IP, por tanto solo necesita establecer una trayectoria para el envió y la recepción de los paquetes de voz.

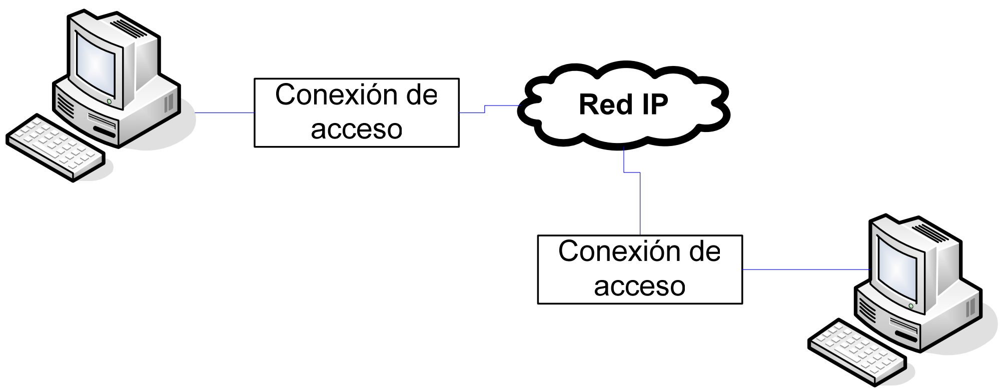


**Figura 15 Comunicación PC a PC**

### 4.3 LLAMADAS PC A TELÉFONO

Otro posible escenario es el de comunicación entre un PC y un teléfono convencional utilizando tanto la red IP como la RTPC. En el caso de que el origen de la llamada sea un PC, la autenticación y autorización de usuario, así como el inter-funcionamiento con la red de señalización SS7 son asuntos manejados directamente por el softswitch. Si la llamada la origina un teléfono convencional, se requiere traducir el número marcado (en formato E.164) a la dirección IP del computador llamado en la red, el resto del proceso es igual al explicado en la sección 4.1.

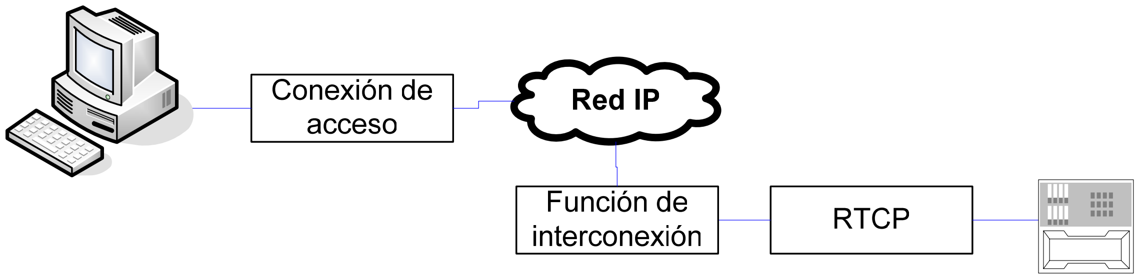


**Figura 16 Comunicación de PC a Teléfono**

### *4.4 GATEWAY CORPORATIVO/CPE*

Cuando un extremo de la llamada es tiene características IP nativas (terminal IP o teléfono con salida IP) el acceso no necesita usar la red de conmutación de circuitos convencional. Para realizar llamadas entre redes diferentes, solo es necesario un mecanismo de traducción de direcciones IP-números telefónicos.

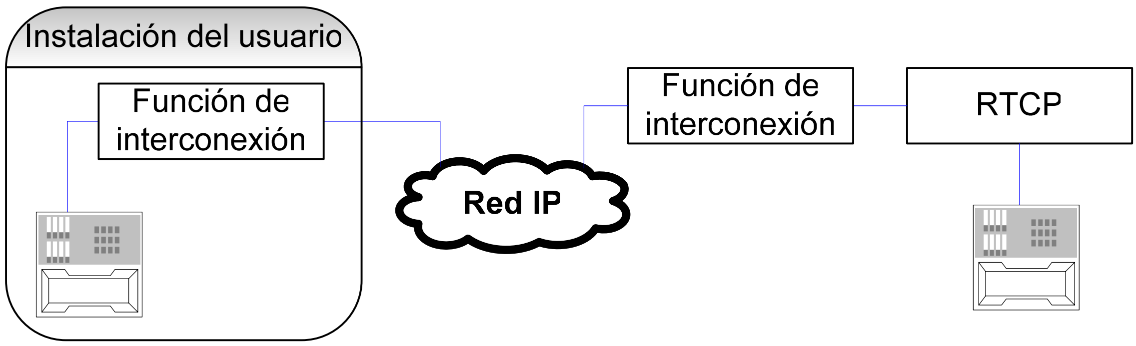


**Figura 17 Comunicación CPE a Teléfono**

# **Capítulo 5**
# MIGRACIÓN HACIA PAQUETES DE VOZ EN TELEFONÍA LOCAL

*Conocer los diferentes esquemas de migración para implantar soluciones que apliquen la tecnología Softswitch sobre redes de telefonía existentes en la actualidad.*

## 5. MIGRACIÓN HACIA PAQUETES DE VOZ EN TELEFONÍA LOCAL.

Hoy en día las soluciones XDSL ofrecen un acceso de banda ancha a numerosos clientes de las centrales de conmutación tradicionales, estas soluciones proporcionan las bases para la migración hacia sistemas de telefonía local basados en la tecnología softswitch. Las soluciones actuales de XDSL permiten el transporte de múltiples líneas telefónicas sobre una sola conexión de banda ancha. Estas soluciones consisten en dos elementos, un IAD[43] localizado en la ubicación del cliente que paquetiza la voz y realiza transmisiones de datos hacia la central, y un gateway de acceso (GW), localizado en la red del proveedor del servicio que convierte la voz paquetizada en un formato que se ajuste a la conmutación de circuitos utilizada en la central local.

En este enfoque, no hay conmutación del tráfico de voz dentro de la red de banda ancha; la primera operación de conmutación que experimenta el tráfico de voz dentro de la red del proveedor del servicio se da en la central de conmutación local, la cual esta *después* del gateway de acceso. Aunque cada IAD pueda tener conexiones con múltiples gateways, cada puerto de un IAD debe estar lógicamente atado a una central de conmutación local. Esta relación estática entre los puertos del IAD y las centrales de conmutación es esencial para la entrega de servicios de telefonía local.

### *5.1 TRANSFORMACIÓN DEL GATEWAY DE ACCESO.*

Gracias a que los GW de VoDSL fueron diseñados con la flexibilidad arquitectural suficiente, es posible agregarles soporte para el control de un MG mediante Megaco, en este caso, el GW se convierte en un verdadero MG. Al acompañar esto con un MGC adecuado, que soporte características de telefonía local, entonces es posible entregar servicios de telefonía local sobre conexiones XDSL sin la necesidad de una central telefónica convencional.

[43] Integrated Access Device.

Esta transformación del GW en un MG puede ser llevada a cabo sin ningún cambio en el protocolo utilizado entre el GW y el IAD para los paquetes de voz. Las nuevas capacidades ofrecidas por el MG son transparentes a los IDAs, que no se dan cuenta que los tonos son generados en el MG al que están conectados, en lugar de la central telefónica. Para funcionar como un MG que soporte los IAD existentes el GW requiere recursos de hardware adicional de procesamiento de señales para poder ejecutar las siguientes funciones.

1. Generación de tonos de progreso de llamada, como tono de marcado, ocupado, etc.
2. Generación de transmisiones de datos dentro de la banda de voz, como aquellos necesitados para enviar la información del identificador de llamadas.
3. Detección y almacenamiento de los dígitos **DTMF** marcados.

El GW esta conectado con la RTPC mediante conexiones basadas en circuitos que soporten un protocolo de acceso a la red como GR-303 o V5.2. El MG transformado puede hacer uso de las mismas interfaces físicas, solo que estas están conectadas ahora a un conmutador tandem y son controlados por protocolos de señalización como SS7. Idealmente el MG tendrá la habilidad de terminar  el enlace de datos físico y se conectara con el MGC enviando mensajes sobre una red IP utilizando un protocolo estándar como SCTP.

## *5.2 MGs  SOPORTANDO ENCAMINAMIENTO DE PAQUETES*

El MG descrito en la sección anterior soporta accesos de paquetes de voz (como VoDSL) con una conversión a encaminamiento basado en circuitos. Esta clase de funcionalidad es un requerimiento absoluto para las soluciones softswitch de telefonía local, debido a que mucho del trafico que se origina en las IAD terminara en la misma área de llamadas o en una conexión **POTS** común, administrada por una central de conmutación local convencional basada en circuitos. Por tanto, una gran parte del tráfico que llega al MG en el lado de acceso tendrá que ser entregado a un conmutador tandem de acceso local mediante un encaminamiento de conmutación de circuitos, típicamente un SS7 IMT[44].

En la figura 18 se aprecian varios casos de acceso desde IADs.

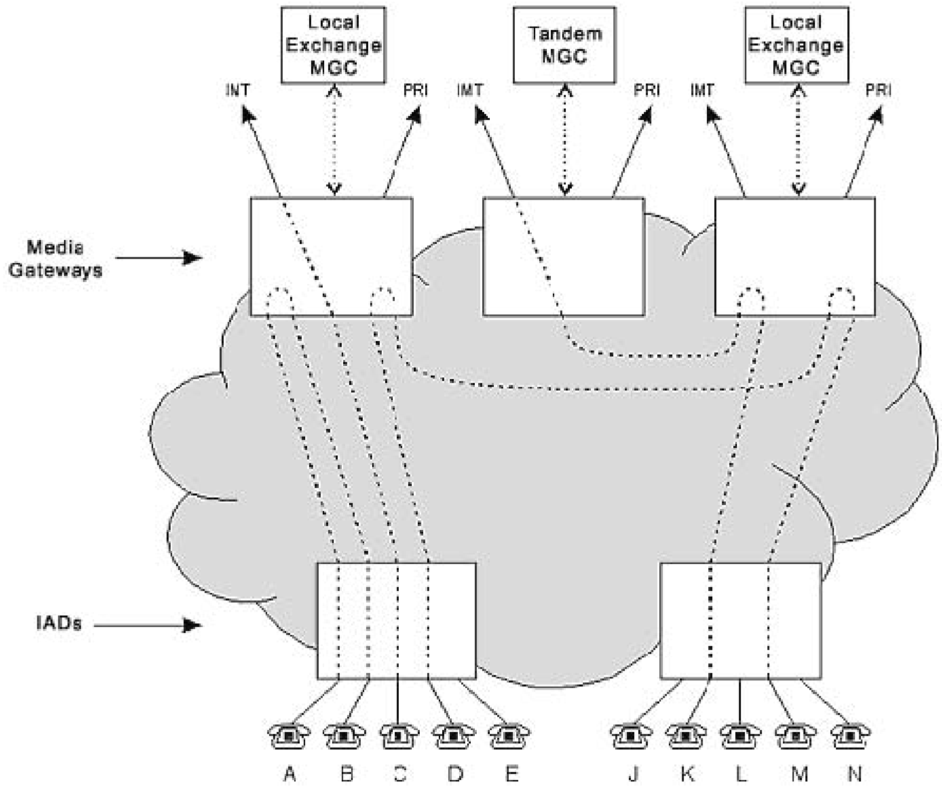

**Figura 18 Casos de acceso desde IADs**

1. El teléfono C hace una llamada local con entrega a otro proveedor de servicio.
2. El teléfono K hace una llamada a través de un gateway tandem con entrega a otro proveedor de servicio.
3. El teléfono A hace una llamada a un teléfono en la misma central (teléfono B)
4. El teléfono D hace una llamada al teléfono M, el cual esta en otro IAD.

No pasara mucho tiempo sin embargo, antes de que el encaminamiento de la voz basado en paquetes sea necesario en este escenario. El encaminamiento de paquetes puede ser usado para colocar en red múltiples MGs en la misma área local de llamadas, para proveer una clase de solución de conmutación

[44] Inter-Machine Trunk

local distribuida, o pueden ser usados para entregar el tráfico de voz a proveedores de servicio ubicados a largas distancias.

En cualquier caso, el MG necesitara ser capas de enrutar el trafico de voz del lado de acceso a conmutadores salientes de paquetes de voz. Si la red de acceso y la de conmutación usan la misma tecnología de paquetes de voz, entonces los MG actúan simplemente como conmutadores de paquetes para mover estos entre las redes de acceso y conmutación. Si se utilizan tecnologías diferentes, entonces el MG tendrá que realizar una conversión de protocolos.

Para que los GW de VoDSL migren de forma exitosa a este papel de conmutación de paquetes de voz, deben tener un alto desempeño y un sistema de conmutación de paquetes tolerante a fallas. SI el GW esta diseñado para convertir el tráfico entrante de paquetes de voz, no podrá manejar el traspaso de los paquetes de voz sin introducir un ciclo adicional de depaquetización y repaquetización. Esto es indeseable debido a que agrega retrasos de transmisión e incrementa los costos de implementación.

### *5.3 MIGRANDO EL MG HACIA EL LÍMITE DE LA RED*

La transformación del GW en un MG como el descrito anteriormente traerá la mayoría de las ventajas de un enfoque de softswitch de telefonía local; sin embargo, lo hace sin hacer ningún cambio sobre los IADs que le dan el acceso al cliente a la red de paquetes. La combinación de MG y MGC permitirán a los proveedores ofrecer servicios de telefonía locales a una fracción del costo de una central de conmutación típica.

La economía asociada con esta clase de soluciones es tal que los portadores pueden atender mercados locales con solo unos cuantos cientos de líneas con ganancias significativas. Ellos podrán atraer clientes combinando servicios de voz y datos con innovaciones y características específicas para cada segmento, en lugar de confiar solo en los grandes descuentos en los servicios de voz.

Pero hay un nivel de  optimización aun mayor que puede ser útil para ciertos tipos de clientes. La solución descrita hasta ahora confía en un MG localizado en la red del proveedor del servicio, que de el tono de marcado; como resultado de esto cada puerto de los IAD debe estar atado permanentemente a un MG especifico en la red del proveedor.

La consecuencia de esto es que todo el tráfico de un puerto de un IAD dado, tiene que pasar físicamente a través del MG que lo controla. Esto no es necesariamente algo malo, el MG provee esencialmente funciones de conversión de medios e la mayoría de las llamadas para hacer la entrega a las conexiones de conmutación de circuitos locales y de larga distancia; y para realizar la conversión de protocolos de voz cuando sea necesario.

### 5.3.1 El IAD como MG

El modelo funcional que se ha explicado hasta el momento asume que no existe ninguna clase de conmutación entre el IAD y el MG del proveedor de servicio. Para alejarse de ese modelo, se pueden mover algunas de las funcionalidades de los MG a los IAD y establecer una conexión MEGACO entre el IAD y el MGC. EL MGC aun debe estar físicamente en la red del proveedor debido a que debe conectarse a la red SS7 y dar soporte a llamadas sobre la RPTC y porque no es factible para las PYMES o los usuarios residenciales tener su propio MGC con conexiones SS7.

La funcionalidad que seria necesaria agregar a los IAD compromete básicamente hardware de procesamiento de señales para realizar la generación y reconocimiento de tonos; así como soporte para el protocolo MEGACO. Estas adiciones permitirían a los IAD convertirse en MG que les ofrecerían funcionalidades de conmutación a los usuarios finales. Este nuevo dispositivo soportaría puertos para telefonía análoga y puertos de acceso para paquetes de voz (típicamente sobre conexiones DSL) y a diferencia de lo explicado anteriormente no tiene una conexión fija con un MG dado.

Tiene libertad de establecer conexiones con cualquier dispositivo que tenga los mismos protocolos, esta libertar esta controlada por el MGC, el cual es responsable de analizar los dígitos de marcado almacenados por el IAD/MG y de determinar el enrutamiento de la llamada. Por ejemplo, si una llamada debe terminar en una central de conmutación de otro operador en la misma área; el MGC dará instrucciones al IAD/MG para establecer una conexión con el MG apropiado en la red del otro operador, donde se le hará el procesamiento respectivo. Para que este modelo opere satisfactoriamente debe ser posible para un IAD/MG establecer conexiones con MG arbitrarios a voluntad.

Se considera la opción de darle tales capacidades a los IAD porque ocasionalmente se vuelve indeseable que todo el tráfico originado en un IAD pase a través del mismo MG en la red del proveedor. Dos de tales circunstancias se explican a continuación.

### 5.3.2 Conexiones de paquetes de voz directas entre IADs.

Un cliente puede tener varias sucursales que tienen acceso para paquetes de voz, donde todos los IADs están conectados a una red de paquetes común. Si hubiese una cantidad substancial de tráfico intra-empresarial entre estos IADs, seria mas eficiente que se establecieran las conexiones directamente entre los IADs que con un MG en el proveedor. Un proveedor puede escoger dar soporte a esta clase de escenario con una solución VPN de voz; en este caso el MGC podría ser configurado para manejar las llamadas entre comunidades de IAD/MG utilizando marcaciones abreviadas.

### 5.3.3 Conexiones directas de paquetes de voz a gateways de conmutación remotos.

A medida que las redes de acceso para paquetes de voz se vuelven ubicuas, debe ser posible para un IAD/MG realizar una conexión directa con GW remotos que se encuentren tal vez en redes de larga distancia o en redes de paquetes de voz internacionales. Si las redes de acceso y la de larga distancia son manejadas por proveedores diferentes, solo se necesitarían acuerdos para utilizar los mismos protocolos y hacer este escenario posible. Esto permitiría a empresas multinacionales establecer sistemas de comunicación entre sus diferentes sedes de forma mucho más eficiente.

En la figura 19 se pueden apreciar varios esquemas de aplicación de IADs con capacidades de MG.

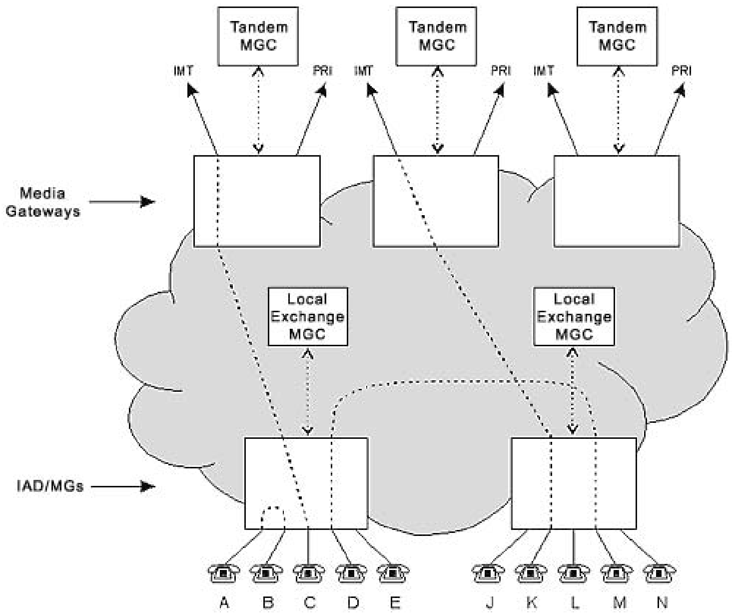


**Figura 19 Casos de acceso desde IAD/MG**

1. El teléfono C hace una llamada local a otro operador.
2. El teléfono K hace una llamada a través de un GW de un tandem remoto con entrega a otro proveedor de servicio.
3. El teléfono A hace una llamada dentro de la misma sede al teléfono B.
4. El teléfono D hace una llamada al teléfono M, el cual esta en otro IAD.

Aquí se puede apreciar claramente que la implementación de esquemas que utilicen IAD/MG, tienen la clara ventaja de reducir el consumo de recursos de la red del proveedor y dar mayores libertades para la generación de servicios de valor agregado a los clientes.

# 6 CONCLUSIONES

Las redes de paquetes se están convirtiendo en el medio común de transmisión de datos para la mayoría de las aplicaciones. El futuro de las telecomunicaciones será ampliamente afectado por el desarrollo de nuevas y más poderosas herramientas para las redes de paquetes, de forma que estas se puedan extender y absorber a las demás redes para así poder crear una gran red donde converjan todos los servicios que se puedan ofrecer.

Con lo expuesto en este trabajo se puede concluir que el desarrollo y la implementación de la tecnología Softswitch, tendrán un impacto muy significativo en el objetivo de crear una sola gran red convergente; al permitir la integración de los servicios más utilizados en la actualidad, la voz, datos y el video. Esto da varios pasos hacia delante para la integración de las redes y los clientes de muchos países (Europa y USA principalmente) han comenzado a sentir las grandes diferencias que ofrece el softswitch, sin embargo se han presentado muchos obstáculos en el camino.

A pesar de que la implementación de esta clase de tecnologías representaría un gran ahorro en muchos aspectos, como se ha mencionado antes, pero la mayoría de los proveedores aun no implementan estas tecnologías de forma masiva, lo cual ha retrasado considerablemente la expansión y la explotación de las capacidades de la tecnología. La razón principal para que esto este sucediendo es que la mayoría de los proveedores ya ha realizado grandes inversiones en la infraestructura existente actualmente, y esa es una inversión que no se puede dejar perder.

En los países del tercer mundo la situación es aun más crítica debido a las profundas crisis económicas que han afectado a estas regiones en los últimos años y que han incrementado aún mas la brecha tecnológica respecto de los países desarrollados. Esto ha afectado notablemente la implementación de nuevas tecnologías en estos países, especialmente tecnologías como la Softswitch, que es relativamente costosa de implementar.

Estos factores, completamente ajenos al desarrollo tecnológico como tal, han impedido que el ritmo de crecimiento de esta tecnología llegue hasta niveles deseables para el establecimiento de redes convergentes globales, además de que limita la cantidad de nodos de la red que pueden ofrecer las características mínimas de calidad y disponibilidad exigidas para la mayoría de los servicios que la tecnología softswitch puede ofrecer.

# 7 BIBLIOGRAFÍA


1. Frank Ohrtman, Softswitch : Architecture for VoIP. Ed. McGraw-Hill Professional; 1ª edición. 359 Paginas.
2. Faulkner Information, SoftSwitch Technology. Ed. Faulkner Information Services. 1a edición. 250 Paginas.
3. Introduccion a la tecnología Softswitch. http://www.mobilein.com/what_is_softswitch.htm
4. Tutorial de Ericsson para Softswitch http://www.ericsson.com/products/hp/Softswitch_pa.shtml
5. Tutorial del International Engineer Consortium sobre Softswitch http://www.iec.org/online/tutorials/softswitch/topic01.html
6. Tutorial del International Engineer Consortium sobre Media Gateways http://www.iec.org/online/tutorials/media_gateway/topic02.html
7. Javier Ríos: Introducción a la tecnología Softswitch. http://www.monografias.com/trabajos14/softswitch/softswitch.shtml
8. Foros oficiales de H.323 http://www.h323forum.org/papers/
9. RFC3525 – MEGACO http://www.javvin.com/protocol/rfc3525.pdf
10. Draft de MEGACO de la IETF http://www.javvin.com/protocol/megaco-h248v2.pdf

# ANEXOS

## RFC2960

Network Working Group
Request for Comments: 2960
Category: Standards Track

R. Stewart
Q. Xie
Motorola
K. Morneault
C. Sharp
Cisco
H. Schwarzbauer
Siemens
T. Taylor
Nortel Networks
I. Rytina
Ericsson
M. Kalla
Telcordia
L. Zhang
UCLA
V. Paxson
ACIRI
October 2000

Stream Control Transmission Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice



Abstract

This document describes the Stream Control Transmission Protocol (SCTP). SCTP is designed to transport PSTN signaling messages over IP networks, but is capable of broader applications.

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users:

-- acknowledged error-free non-duplicated transfer of user data,
-- data fragmentation to conform to discovered path MTU size,

-- sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages,

-- optional bundling of multiple user messages into a single SCTP packet, and

-- network-level fault tolerance through supporting of multi-homing at either or both ends of an association.

The design of SCTP includes appropriate congestion avoidance behavior and resistance to flooding and masquerade attacks.

Table of Contents

1. Introduction

This section explains the reasoning behind the development of the Stream Control Transmission Protocol (SCTP), the services it offers, and the basic concepts needed to understand the detailed description of the protocol.

1.1 Motivation

TCP [<A HREF="/rfcs/rfc793.html">RFC793</A>] has performed immense service as the primary means of reliable data transfer in IP networks. However, an increasing number of recent applications have found TCP too limiting, and have incorporated their own reliable data transfer protocol on top of UDP [<A HREF="/rfcs/rfc768.html">RFC768</A>]. The limitations which users have wished to bypass include the following:

-- TCP provides both reliable data transfer and strict order-of-transmission delivery of data. Some applications need reliable transfer without sequence maintenance, while others would be satisfied with partial ordering of the data. In both of these cases the head-of-line blocking offered by TCP causes unnecessary delay.

-- The stream-oriented nature of TCP is often an inconvenience. Applications must add their own record marking to delineate their messages, and must make explicit use of the push facility to ensure that a complete message is transferred in a reasonable time.

-- The limited scope of TCP sockets complicates the task of providing highly-available data transfer capability using multi-homed hosts.

-- TCP is relatively vulnerable to denial of service attacks, such as SYN attacks.

Transport of PSTN signaling across the IP network is an application for which all of these limitations of TCP are relevant. While this application directly motivated the development of SCTP, other applications may find SCTP a good match to their requirements.

## 1.2 Architectural View of SCTP

SCTP is viewed as a layer between the SCTP user application ("SCTP user" for short) and a connectionless packet network service such as IP. The remainder of this document assumes SCTP runs on top of IP. The basic service offered by SCTP is the reliable transfer of user messages between peer SCTP users. It performs this service within the context of an association between two SCTP endpoints. Section 10 of this document sketches the API which should exist at the boundary between the SCTP and the SCTP user layers.

SCTP is connection-oriented in nature, but the SCTP association is a broader concept than the TCP connection. SCTP provides the means for each SCTP endpoint (Section 1.4) to provide the other endpoint

(during association startup) with a list of transport addresses (i.e., multiple IP addresses in combination with an SCTP port) through which that endpoint can be reached and from which it will originate SCTP packets. The association spans transfers over all of the possible source/destination combinations which may be generated from each endpoint's lists.

```
      |  SCTP User  |                                    |  SCTP User
|
```

```
      | Application |                                    | Application
|
      |-------------|                                    |------------
-|
      |    SCTP     |                                    |    SCTP
|
      |  Transport  |                                    |  Transport
|
      |   Service   |                                    |   Service
|
      |-------------|                                    |------------
-|
      |             |One or more    ----      One or more|
|
      | IP Network  |IP address      \/        IP address| IP Network
|
      |   Service   |appearances     /\       appearances|   Service
|
      |_____________|               ----
|_____________|

        SCTP Node A |&lt;-------- Network transport -------&gt;| SCTP
Node B
```

Figure 1: An SCTP Association

1.3 Functional View of SCTP

The SCTP transport service can be decomposed into a number of functions.  These are depicted in Figure 2 and explained in the remainder of this section.

```
                          SCTP User Application

         -----------------------------------------------------
          _____________                ____________________
         |             |              | Sequenced delivery |
         | Association |              |   within streams   |
         |             |              |____________________|
         |   startup   |
         |             |       ____________________________
         |     and     |      |    User Data Fragmentation |
         |             |      |____________________________|
         |   takedown  |
         |             |       ____________________________
         |             |      |     Acknowledgement        |
         |             |      |          and               |
         |             |      |    Congestion Avoidance    |
         |             |      |____________________________|
         |             |
         |             |       ____________________________
         |             |      |        Chunk Bundling      |
         |             |      |____________________________|
         |             |
         |             |   ________________________________
         |             |  |      Packet Validation         |
         |             |  |________________________________|
         |             |
         |             |   ________________________________
         |             |  |     Path Management            |
         |_____________|  |________________________________|
```

Figure 2: Functional View of the SCTP Transport Service

### 1.3.1 Association Startup and Takedown

An association is initiated by a request from the SCTP user (see the description of the ASSOCIATE (or SEND) primitive in Section 10).

A cookie mechanism, similar to one described by Karn and Simpson in [<A HREF="/rfcs/rfc2522.html">RFC2522</A>], is employed during the initialization to provide protection against security attacks. The cookie mechanism uses a four-way handshake, the last two legs of which are allowed to carry user data for fast setup. The startup sequence is described in Section 5 of this document.

SCTP provides for graceful close (i.e., shutdown) of an active association on request from the SCTP user. See the description of the SHUTDOWN primitive in Section 10. SCTP also allows ungraceful close (i.e., abort), either on request from the user (ABORT

primitive) or as a result of an error condition detected within the SCTP layer. Section 9 describes both the graceful and the ungraceful close procedures.

SCTP does not support a half-open state (like TCP) wherein one side may continue sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of the graceful close (see Section 9).

### 1.3.2 Sequenced Delivery within Streams

The term "stream" is used in SCTP to refer to a sequence of user messages that are to be delivered to the upper-layer protocol in order with respect to other messages within the same stream. This is in contrast to its usage in TCP, where it refers to a sequence of bytes (in this document a byte is assumed to be eight bits).

The SCTP user can specify at association startup time the number of streams to be supported by the association. This number is negotiated with the remote end (see Section 5.1.1). User messages are associated with stream numbers (SEND, RECEIVE primitives, Section 10). Internally, SCTP assigns a stream sequence number to each message passed to it by the SCTP user. On the receiving side, SCTP ensures that messages are delivered to the SCTP user in sequence within a given stream. However, while one stream may be blocked waiting for the next in-sequence user message, delivery from other streams may proceed.

SCTP provides a mechanism for bypassing the sequenced delivery service. User messages sent using this mechanism are delivered to the SCTP user as soon as they are received.

### 1.3.3 User Data Fragmentation

When needed, SCTP fragments user messages to ensure that the SCTP packet passed to the lower layer conforms to the path MTU. On receipt, fragments are reassembled into complete messages before being passed to the SCTP user.

1.3.4 Acknowledgement and Congestion Avoidance

SCTP assigns a Transmission Sequence Number (TSN) to each user data fragment or unfragmented message. The TSN is independent of any stream sequence number assigned at the stream level. The receiving

end acknowledges all TSNs received, even if there are gaps in the sequence. In this way, reliable delivery is kept functionally separate from sequenced stream delivery.

The acknowledgement and congestion avoidance function is responsible for packet retransmission when timely acknowledgement has not been received. Packet retransmission is conditioned by congestion avoidance procedures similar to those used for TCP. See Sections 6 and 7 for a detailed description of the protocol procedures associated with this function.

1.3.5 Chunk Bundling

As described in Section 3, the SCTP packet as delivered to the lower layer consists of a common header followed by one or more chunks. Each chunk may contain either user data or SCTP control information. The SCTP user has the option to request bundling of more than one user messages into a single SCTP packet. The chunk bundling function of SCTP is responsible for assembly of the complete SCTP packet and its disassembly at the receiving end.

During times of congestion an SCTP implementation MAY still perform bundling even if the user has requested that SCTP not bundle. The user's disabling of bundling only affects SCTP implementations that may delay a small period of time before transmission (to attempt to encourage bundling). When the user layer disables bundling, this small delay is prohibited but not bundling that is performed during congestion or retransmission.

1.3.6 Packet Validation

A mandatory Verification Tag field and a 32 bit checksum field (see Appendix B for a description of the Adler-32 checksum) are included in the SCTP common header. The Verification Tag value is chosen by each end of the association during association startup. Packets received without the expected Verification Tag value are discarded, as a protection against blind masquerade attacks and against stale SCTP packets from a previous association. The Adler-32 checksum should be set by the sender of each SCTP packet to provide additional protection against data corruption in the network. The receiver of an SCTP packet with an invalid Adler-32 checksum silently discards the packet.

1.3.7 Path Management

The sending SCTP user is able to manipulate the set of transport addresses used as destinations for SCTP packets through the primitives described in Section 10. The SCTP path management function chooses the destination transport address for each outgoing SCTP packet based on the SCTP user's instructions and the currently perceived reachability status of the eligible destination set. The path management function monitors reachability through heartbeats when other packet traffic is inadequate to provide this information and advises the SCTP user when reachability of any far-end transport address changes. The path management function is also responsible for reporting the eligible set of local transport addresses to the far end during association startup, and for reporting the transport addresses returned from the far end to the SCTP user.

At association start-up, a primary path is defined for each SCTP endpoint, and is used for normal sending of SCTP packets.

On the receiving end, the path management is responsible for verifying the existence of a valid SCTP association to which the inbound SCTP packet belongs before passing it for further processing.

Note: Path Management and Packet Validation are done at the same time, so although described separately above, in reality they cannot be performed as separate items.

## 1.4 Key Terms

Some of the language used to describe SCTP has been introduced in the previous sections. This section provides a consolidated list of the key terms and their definitions.

- o Active destination transport address: A transport address on a peer endpoint which a transmitting endpoint considers available for receiving user messages.

- o Bundling: An optional multiplexing operation, whereby more than one user message may be carried in the same SCTP packet. Each user message occupies its own DATA chunk.

- o Chunk: A unit of information within an SCTP packet, consisting of a chunk header and chunk-specific content.

- o Congestion Window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

- o Cumulative TSN Ack Point: The TSN of the last DATA chunk acknowledged via the Cumulative TSN Ack field of a SACK.

- o Idle destination address: An address that has not had user messages sent to it within some length of time, normally the HEARTBEAT interval or greater.

- Inactive destination transport address: An address which is considered inactive due to errors and unavailable to transport user messages.

- Message = user message:  Data submitted to SCTP by the Upper Layer Protocol (ULP).

- Message Authentication Code (MAC):  An integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties.  In SCTP it is used by an endpoint to validate the State Cookie information that is returned from the peer in the COOKIE ECHO chunk.  The term "MAC" has different meanings in different contexts.  SCTP uses this term with the same meaning as in [<A HREF="/rfcs/rfc2104.html">RFC2104</A>].

- Network Byte Order: Most significant byte first, a.k.a., Big Endian.

- Ordered Message: A user message that is delivered in order with respect to all previous user messages sent within the stream the message was sent on.

- Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

- Path: The route taken by the SCTP packets sent by one SCTP endpoint to a specific destination transport address of its peer SCTP endpoint.  Sending to different destination transport addresses does not necessarily guarantee getting separate paths.

- Primary Path: The primary path is the destination and source address that will be put into a packet outbound to the peer endpoint by default.  The definition includes the source address since an implementation MAY wish to specify both destination and source address to better control the return path taken by reply chunks and on which interface the packet is transmitted when the data sender is multi-homed.

- Receiver Window (rwnd): An SCTP variable a data sender uses to store the most recently calculated receiver window of its peer, in number of bytes.  This gives the sender an indication of the space available in the receiver's inbound buffer.

- SCTP association: A protocol relationship between SCTP endpoints, composed of the two SCTP endpoints and protocol state information including Verification Tags and the currently active set of Transmission Sequence Numbers (TSNs), etc.  An association can be uniquely identified by the transport addresses used by the endpoints in the association.  Two SCTP endpoints MUST NOT have more than one SCTP association between them at any given time.

o SCTP endpoint: The logical sender/receiver of SCTP packets. On a
multi-homed host, an SCTP endpoint is represented to its peers as
a combination of a set of eligible destination transport addresses
to which SCTP packets can be sent and a set of eligible source
transport addresses from which SCTP packets can be received. All
transport addresses used by an SCTP endpoint must use the same
port number, but can use multiple IP addresses. A transport
address used by an SCTP endpoint must not be used by another SCTP
endpoint. In other words, a transport address is unique to an
SCTP endpoint.

o SCTP packet (or packet): The unit of data delivery across the
interface between SCTP and the connectionless packet network
(e.g., IP). An SCTP packet includes the common SCTP header,
possible SCTP control chunks, and user data encapsulated within
SCTP DATA chunks.

o SCTP user application (SCTP user): The logical higher-layer
application entity which uses the services of SCTP, also called
the Upper-layer Protocol (ULP).

o Slow Start Threshold (ssthresh): An SCTP variable. This is the
threshold which the endpoint will use to determine whether to
perform slow start or congestion avoidance on a particular
destination transport address. Ssthresh is in number of bytes.

o Stream: A uni-directional logical channel established from one to
another associated SCTP endpoint, within which all user messages
are delivered in sequence except for those submitted to the
unordered delivery service.

Note: The relationship between stream numbers in opposite directions
is strictly a matter of how the applications use them. It is the
responsibility of the SCTP user to create and manage these
correlations if they are so desired.

o Stream Sequence Number: A 16-bit sequence number used internally
by SCTP to assure sequenced delivery of the user messages within a
given stream. One stream sequence number is attached to each user
message.

o Tie-Tags: Verification Tags from a previous association. These
Tags are used within a State Cookie so that the newly restarting
association can be linked to the original association within the
endpoint that did not restart.

o Transmission Control Block (TCB): An internal data structure
created by an SCTP endpoint for each of its existing SCTP
associations to other SCTP endpoints. TCB contains all the status

and operational information for the endpoint to maintain and manage the corresponding association.

- o Transmission Sequence Number (TSN): A 32-bit sequence number used internally by SCTP. One TSN is attached to each chunk containing user data to permit the receiving SCTP endpoint to acknowledge its receipt and detect duplicate deliveries.

- o Transport address: A Transport Address is traditionally defined by Network Layer address, Transport Layer protocol and Transport Layer port number. In the case of SCTP running over IP, a transport address is defined by the combination of an IP address and an SCTP port number (where SCTP is the Transport protocol).

- o Unacknowledged TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) which has been received by the endpoint but for which an acknowledgement has not yet been sent. Or in the opposite case, for a packet that has been sent but no acknowledgement has been received.

- o Unordered Message: Unordered messages are "unordered" with respect to any other message, this includes both other unordered messages as well as other ordered messages. Unordered message might be delivered prior to or later than ordered messages sent on the same stream.

- o User message: The unit of data delivery across the interface between SCTP and its user.

- o Verification Tag: A 32 bit unsigned integer that is randomly generated. The Verification Tag provides a key that allows a receiver to verify that the SCTP packet belongs to the current association and is not an old or stale packet from a previous association.

1.5. Abbreviations

MAC - Message Authentication Code [<A HREF="/rfcs/rfc2104.html">RFC2104</A>]

RTO - Retransmission Time-out

RTT - Round-trip Time

RTTVAR - Round-trip Time Variation

SCTP - Stream Control Transmission Protocol

SRTT - Smoothed RTT

TCB - Transmission Control Block

TLV    - Type-Length-Value Coding Format

TSN    - Transmission Sequence Number

ULP    - Upper-layer Protocol

## 1.6 Serial Number Arithmetic

It is essential to remember that the actual Transmission Sequence Number space is finite, though very large.  This space ranges from 0 to 2**32 - 1. Since the space is finite, all arithmetic dealing with Transmission Sequence Numbers must be performed modulo 2**32.  This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2**32 - 1 to 0 again.  There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values.  When referring to TSNs, the symbol "=<" means "less than or equal"(modulo 2**32).

Comparisons and arithmetic on TSNs in this document SHOULD use Serial Number Arithmetic as defined in [<A HREF="/rfcs/rfc1982.html">RFC1982</A>] where SERIAL_BITS = 32.

An endpoint SHOULD NOT transmit a DATA chunk with a TSN that is more than 2**31 - 1 above the beginning TSN of its current send window. Doing so will cause problems in comparing TSNs.

Transmission Sequence Numbers wrap around when they reach 2**32 - 1. That is, the next TSN a DATA chunk MUST use after transmitting TSN = 2*32 - 1 is TSN = 0.

Any arithmetic done on Stream Sequence Numbers SHOULD use Serial Number Arithmetic as defined in [<A HREF="/rfcs/rfc1982.html">RFC1982</A>] where SERIAL_BITS = 16.

All other arithmetic and comparisons in this document uses normal arithmetic.

## 2. Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [<A HREF="/rfcs/rfc2119.html">RFC2119</A>].

## 3.  SCTP packet Format

An SCTP packet is composed of a common header and chunks. A chunk contains either control information or user data.

The SCTP packet format is shown below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                        Common Header
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                          Chunk #1
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                           ...
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                          Chunk #n
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Multiple chunks can be bundled into one SCTP packet up to the MTU size, except for the INIT, INIT ACK, and SHUTDOWN COMPLETE chunks. These chunks MUST NOT be bundled with any other chunk in a packet. See Section 6.10 for more details on chunk bundling.

If a user data message doesn't fit into one SCTP packet it can be fragmented into multiple chunks using the procedure defined in Section 6.9.

All integer fields in an SCTP packet MUST be transmitted in network byte order, unless otherwise stated.

## 3.1 SCTP Common Header Field Descriptions

SCTP Common Header Format

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Source Port Number        |     Destination Port Number
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                      Verification Tag
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                           Checksum
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Source Port Number: 16 bits (unsigned integer)

This is the SCTP sender's port number.  It can be used by the receiver in combination with the source IP address, the SCTP destination port and possibly the destination IP address to identify the association to which this packet belongs.

Destination Port Number: 16 bits (unsigned integer)

This is the SCTP port number to which this packet is destined. The receiving host will use this port number to de-multiplex the SCTP packet to the correct receiving endpoint/application.

Verification Tag: 32 bits (unsigned integer)

The receiver of this packet uses the Verification Tag to validate the sender of this SCTP packet. On transmit, the value of this Verification Tag MUST be set to the value of the Initiate Tag received from the peer endpoint during the association initialization, with the following exceptions:

- A packet containing an INIT chunk MUST have a zero Verification Tag.
- A packet containing a SHUTDOWN-COMPLETE chunk with the T-bit set MUST have the Verification Tag copied from the packet with the SHUTDOWN-ACK chunk.
- A packet containing an ABORT chunk may have the verification tag copied from the packet which caused the ABORT to be sent. For details see Section 8.4 and 8.5.

An INIT chunk MUST be the only chunk in the SCTP packet carrying it.

Checksum: 32 bits (unsigned integer)

This field contains the checksum of this SCTP packet. Its calculation is discussed in Section 6.8. SCTP uses the Adler-32 algorithm as described in Appendix B for calculating the checksum

3.2 Chunk Field Descriptions

The figure below illustrates the field format for the chunks to be transmitted in the SCTP packet. Each chunk is formatted with a Chunk Type field, a chunk-specific Flag field, a Chunk Length field, and a Value field.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Chunk Type  | Chunk  Flags  |        Chunk Length           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                          Chunk Value                          /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Type: 8 bits (unsigned integer)

This field identifies the type of information contained in the Chunk Value field. It takes a value from 0 to 254. The value of 255 is reserved for future use as an extension field.

The values of Chunk Types are defined as follows:

| ID Value | Chunk Type |
| --- | --- |
| 0 | - Payload Data (DATA) |
| 1 | - Initiation (INIT) |
| 2 | - Initiation Acknowledgement (INIT ACK) |
| 3 | - Selective Acknowledgement (SACK) |
| 4 | - Heartbeat Request (HEARTBEAT) |
| 5 | - Heartbeat Acknowledgement (HEARTBEAT ACK) |
| 6 | - Abort (ABORT) |
| 7 | - Shutdown (SHUTDOWN) |
| 8 | - Shutdown Acknowledgement (SHUTDOWN ACK) |
| 9 | - Operation Error (ERROR) |
| 10 | - State Cookie (COOKIE ECHO) |
| 11 | - Cookie Acknowledgement (COOKIE ACK) |
| 12 | - Reserved for Explicit Congestion Notification Echo (ECNE) |
| 13 | - Reserved for Congestion Window Reduced (CWR) |
| 14 | - Shutdown Complete (SHUTDOWN COMPLETE) |
| 15 to 62 | - reserved by IETF |
| 63 | - IETF-defined Chunk Extensions |
| 64 to 126 | - reserved by IETF |
| 127 | - IETF-defined Chunk Extensions |
| 128 to 190 | - reserved by IETF |
| 191 | - IETF-defined Chunk Extensions |
| 192 to 254 | - reserved by IETF |
| 255 | - IETF-defined Chunk Extensions |

Chunk Types are encoded such that the highest-order two bits specify the action that must be taken if the processing endpoint does not recognize the Chunk Type.

00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.

01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized parameter in an 'Unrecognized Parameter Type' (in either an ERROR or in the INIT ACK).

10 - Skip this chunk and continue processing.

11 - Skip this chunk and continue processing, but report in an ERROR Chunk using the 'Unrecognized Chunk Type' cause of error.

Note: The ECNE and CWR chunk types are reserved for future use of Explicit Congestion Notification (ECN).

Chunk Flags: 8 bits

The usage of these bits depends on the chunk type as given by the Chunk Type. Unless otherwise specified, they are set to zero on transmit and are ignored on receipt.

Chunk Length: 16 bits (unsigned integer)

This value represents the size of the chunk in bytes including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields. Therefore, if the Chunk Value field is zero-length, the Length field will be set to 4. The Chunk Length field does not count any padding.

Chunk Value: variable length

The Chunk Value field contains the actual information to be transferred in the chunk. The usage and format of this field is dependent on the Chunk Type.

The total length of a chunk (including Type, Length and Value fields) MUST be a multiple of 4 bytes. If the length of the chunk is not a multiple of 4 bytes, the sender MUST pad the chunk with all zero bytes and this padding is not included in the chunk length field. The sender should never pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

SCTP defined chunks are described in detail in Section 3.3. The guidelines for IETF-defined chunk extensions can be found in Section 13.1 of this document.

3.2.1  Optional/Variable-length Parameter Format

Chunk values of SCTP control chunks consist of a chunk-type-specific header of required fields, followed by zero or more parameters. The optional and variable-length parameters contained in a chunk are defined in a Type-Length-Value format as shown below.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Parameter Type       |       Parameter Length        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    \                                                               \
    /                       Parameter Value                         /
    \                                                               \
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Parameter Type: 16 bits (unsigned integer)

The Type field is a 16 bit identifier of the type of parameter. It takes a value of 0 to 65534.

The value of 65535 is reserved for IETF-defined extensions. Values other than those defined in specific SCTP chunk description are reserved for use by IETF.

Chunk Parameter Length: 16 bits (unsigned integer)

The Parameter Length field contains the size of the parameter in bytes, including the Parameter Type, Parameter Length, and Parameter Value fields. Thus, a parameter with a zero-length Parameter Value field would have a Length field of 4. The Parameter Length does not include any padding bytes.

Chunk Parameter Value: variable-length.

The Parameter Value field contains the actual information to be transferred in the parameter.

The total length of a parameter (including Type, Parameter Length and Value fields) MUST be a multiple of 4 bytes. If the length of the parameter is not a multiple of 4 bytes, the sender pads the Parameter at the end (i.e., after the Parameter Value field) with all zero bytes. The length of the padding is not included in the parameter length field. A sender SHOULD NOT pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

The Parameter Types are encoded such that the highest-order two bits specify the action that must be taken if the processing endpoint does not recognize the Parameter Type.

00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.

01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized parameter in an 'Unrecognized Parameter Type' (in either an ERROR or in the INIT ACK).

10 - Skip this parameter and continue processing.

11 - Skip this parameter and continue processing but report the unrecognized parameter in an 'Unrecognized Parameter Type' (in either an ERROR or in the INIT ACK).

The actual SCTP parameters are defined in the specific SCTP chunk sections. The rules for IETF-defined parameter extensions are defined in Section 13.2.

3.3 SCTP Chunk Definitions

This section defines the format of the different SCTP chunk types.

3.3.1 Payload Data (DATA) (0)

The following format MUST be used for the DATA chunk:

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |   Type = 0    | Reserved|U|B|E|    Length
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |                              TSN
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |      Stream Identifier S      |   Stream Sequence Number n
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |                  Payload Protocol Identifier
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       \
\
       /                 User Data (seq n of Stream S)
/
       \
\
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Reserved: 5 bits

Should be set to all '0's and ignored by the receiver.

U bit: 1 bit

The (U)nordered bit, if set to '1', indicates that this is an unordered DATA chunk, and there is no Stream Sequence Number assigned to this DATA chunk.  Therefore, the receiver MUST ignore the Stream Sequence Number field.

After re-assembly (if necessary), unordered DATA chunks MUST be dispatched to the upper layer by the receiver without any attempt to re-order.

If an unordered user message is fragmented, each fragment of the message MUST have its U bit set to '1'.

B bit: 1 bit

The (B)eginning fragment bit, if set, indicates the first fragment of a user message.

E bit: 1 bit

The (E)nding fragment bit, if set, indicates the last fragment of a user message.

An unfragmented user message shall have both the B and E bits set to '1'. Setting both B and E bits to '0' indicates a middle fragment of a multi-fragment user message, as summarized in the following table:

| B E | Description |
|---|---|
| 1 0 | First piece of a fragmented user message |
| 0 0 | Middle piece of a fragmented user message |
| 0 1 | Last piece of a fragmented user message |
| 1 1 | Unfragmented Message |

Table 1: Fragment Description Flags

When a user message is fragmented into multiple chunks, the TSNs are used by the receiver to reassemble the message. This means that the TSNs for each fragment of a fragmented user message MUST be strictly sequential.

Length: 16 bits (unsigned integer)

This field indicates the length of the DATA chunk in bytes from the beginning of the type field to the end of the user data field excluding any padding. A DATA chunk with no user data field will have Length set to 16 (indicating 16 bytes).

TSN : 32 bits (unsigned integer)

This value represents the TSN for this DATA chunk. The valid range of TSN is from 0 to 4294967295 (2**32 - 1). TSN wraps back to 0 after reaching 4294967295.

Stream Identifier S: 16 bits (unsigned integer)

Identifies the stream to which the following user data belongs.

Stream Sequence Number n: 16 bits (unsigned integer)

This value represents the stream sequence number of the following user data within the stream S. Valid range is 0 to 65535.

When a user message is fragmented by SCTP for transport, the same stream sequence number MUST be carried in each of the fragments of the message.

Payload Protocol Identifier: 32 bits (unsigned integer)

This value represents an application (or upper layer) specified protocol identifier. This value is passed to SCTP by its upper layer and sent to its peer. This identifier is not used by SCTP but can be used by certain network entities as well as the peer application to identify the type of information being carried in this DATA chunk. This field must be sent even in fragmented DATA chunks (to make sure it is available for agents in the middle of the network).

The value 0 indicates no application identifier is specified by the upper layer for this payload data.

User Data: variable length

This is the payload user data. The implementation MUST pad the end of the data to a 4 byte boundary with all-zero bytes. Any padding MUST NOT be included in the length field. A sender MUST never add more than 3 bytes of padding.

### 3.3.2 Initiation (INIT) (1)

This chunk is used to initiate a SCTP association between two endpoints. The format of the INIT chunk is shown below:

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |   Type = 1    |  Chunk Flags  |      Chunk Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                         Initiate Tag
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |           Advertised Receiver Window Credit (a_rwnd)
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |  Number of Outbound Streams   |  Number of Inbound Streams
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                          Initial TSN
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      \
\
      /              Optional/Variable-Length Parameters
/
```

```
      \
\
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

The INIT chunk contains the following parameters. Unless otherwise noted, each parameter MUST only be included once in the INIT chunk.

| Fixed Parameters | Status |
|---|---|
| Initiate Tag | Mandatory |
| Advertised Receiver Window Credit | Mandatory |
| Number of Outbound Streams | Mandatory |
| Number of Inbound Streams | Mandatory |
| Initial TSN | Mandatory |

| Variable Parameters | Status | Type Value |
|---|---|---|
| IPv4 Address (Note 1) | Optional | 5 |
| IPv6 Address (Note 1) | Optional | 6 |
| Cookie Preservative | Optional | 9 |
| Reserved for ECN Capable (Note 2) | Optional | 32768 (0x8000) |
| Host Name Address (Note 3) | Optional | 11 |
| Supported Address Types (Note 4) | Optional | 12 |

Note 1: The INIT chunks can contain multiple addresses that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN capable field is reserved for future use of Explicit Congestion Notification.

Note 3: An INIT chunk MUST NOT contain more than one Host Name address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name address parameter is present in the received INIT chunk.

Note 4: This parameter, when present, specifies all the address types the sending endpoint can support. The absence of this parameter indicates that the sending endpoint can support any address type.

The Chunk Flags field in INIT is reserved and all bits in it should be set to 0 by the sender and ignored by the receiver. The sequence of parameters within an INIT can be processed in any order.

Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT (the responding end) records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP packet that the receiver of the INIT transmits within this association.

The Initiate Tag is allowed to have any value except 0. See

Section 5.3.1 for more on the selection of the tag value.

If the value of the Initiate Tag in a received INIT chunk is found to be 0, the receiver MUST treat it as an error and close the association by transmitting an ABORT.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT has reserved in association with this window. During the life of the association this buffer space SHOULD not be lessened (i.e. dedicated buffers taken away from this association); however, an endpoint MAY change the value of a_rwnd it sends in SACK chunks.

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT chunk wishes to create in this association. The value of 0 MUST NOT be used.

Note: A receiver of an INIT with the OS value set to 0 SHOULD abort the association.

Number of Inbound Streams (MIS) : 16 bits (unsigned integer)

Defines the maximum number of streams the sender of this INIT chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

Note: A receiver of an INIT with the MIS value of 0 SHOULD abort the association.

Initial TSN (I-TSN) : 32 bits (unsigned integer)

Defines the initial TSN that the sender will use. The valid range is from 0 to 4294967295. This field MAY be set to the value of the Initiate Tag field.

3.3.2.1 Optional/Variable Length Parameters in INIT

The following parameters follow the Type-Length-Value format as defined in Section 3.2.1. Any Type-Length-Value fields MUST come after the fixed-length fields defined in the previous section.

IPv4 Address Parameter (5)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

```
      |        Type = 5               |      Length = 8
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                        IPv4 Address
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

IPv4 Address: 32 bits (unsigned integer)

Contains an IPv4 address of the sending endpoint. It is binary encoded.

IPv6 Address Parameter (6)

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |            Type = 6           |          Length = 20
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |
|
      |                         IPv6 Address
|
      |
|
      |
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

IPv6 Address: 128 bit (unsigned integer)

Contains an IPv6 address of the sending endpoint. It is binary encoded.

Note: A sender MUST NOT use an IPv4-mapped IPv6 address [<A HREF="/rfcs/rfc2373.html">RFC2373</A>] but should instead use an IPv4 Address Parameter for an IPv4 address.

Combined with the Source Port Number in the SCTP common header, the value passed in an IPv4 or IPv6 Address parameter indicates a transport address the sender of the INIT will support for the association being initiated. That is, during the lifetime of this association, this IP address can appear in the source address field of an IP datagram sent from the sender of the INIT, and can be used as a destination address of an IP datagram sent from the receiver of the INIT.

More than one IP Address parameter can be included in an INIT chunk when the INIT sender is multi-homed. Moreover, a multi-homed endpoint may have access to different types of network, thus

more than one address type can be present in one INIT chunk,
i.e.,
IPv4 and IPv6 addresses are allowed in the same INIT chunk.

If the INIT contains at least one IP Address parameter, then the
source address of the IP datagram containing the INIT chunk and
any additional address(es) provided within the INIT can be used
as
destinations by the endpoint receiving the INIT.  If the INIT
does
not contain any IP Address parameters, the endpoint receiving
the
INIT MUST use the source address associated with the received IP
datagram as its sole destination address for the association.

Note that not using any IP address parameters in the INIT and
INIT-ACK is an alternative to make an association more likely to
work across a NAT box.

Cookie Preservative (9)

The sender of the INIT shall use this parameter to suggest to
the
receiver of the INIT for a longer life-span of the State Cookie.

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |          Type = 9             |          Length = 8
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |         Suggested Cookie Life-span Increment (msec.)
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Suggested Cookie Life-span Increment: 32 bits (unsigned integer)

This parameter indicates to the receiver how much increment in
milliseconds the sender wishes the receiver to add to its
default
cookie life-span.

This optional parameter should be added to the INIT chunk by the
sender when it re-attempts establishing an association with a
peer
to which its previous attempt of establishing the association
failed
due to a stale cookie operation error.  The receiver MAY choose
to
ignore the suggested cookie life-span increase for its own
security
reasons.

Host Name Address (11)

The sender of INIT uses this parameter to pass its Host Name (in
place of its IP addresses) to its peer.  The peer is responsible
for resolving the name.  Using this parameter might make it more

likely for the association to work across a NAT box.

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |          Type = 11            |          Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /                          Host Name
/
      \
\
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Host Name: variable length

This field contains a host name in "host name syntax" per <A
HREF="/rfcs/rfc1123.html">RFC1123</A>
Section 2.1 [<A HREF="/rfcs/rfc1123.html">RFC1123</A>]. The
method for resolving the host name is
out of scope of SCTP.

Note: At least one null terminator is included in the Host Name string and must be included in the length.

Supported Address Types (12)

The sender of INIT uses this parameter to list all the address types it can support.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |          Type = 12            |          Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |        Address Type #1        |        Address Type #2
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |        ......
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6, Hostname = 11).

### 3.3.3 Initiation Acknowledgement (INIT ACK) (2):

The INIT ACK chunk is used to acknowledge the initiation of an SCTP association.

The parameter part of INIT ACK is formatted similarly to the INIT chunk. It uses two extra variable parameters: The State Cookie and the Unrecognized Parameter:

The format of the INIT ACK chunk is shown below:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
   |   Type = 2    |  Chunk Flags  |      Chunk Length
|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
   |                         Initiate Tag
|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
   |              Advertised Receiver Window Credit
|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
   |  Number of Outbound Streams   |  Number of Inbound Streams
|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
   |                          Initial TSN
|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
   \
\
   /              Optional/Variable-Length Parameters
/
   \
\
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT ACK records the value of the Initiate
Tag
parameter.  This value MUST be placed into the Verification Tag field of every SCTP packet that the INIT ACK receiver transmits within this association.

The Initiate Tag MUST NOT take the value 0.  See Section 5.3.1
for
more on the selection of the Initiate Tag value.

If the value of the Initiate Tag in a received INIT ACK chunk is found to be 0, the receiver MUST treat it as an error and close the association by transmitting an ABORT.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT ACK has reserved in association
with
this window.  During the life of the association this buffer
space
SHOULD not be lessened (i.e. dedicated buffers taken away from

this association).

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT ACK chunk wishes to create in this association. The value of 0 MUST NOT be used.

Note: A receiver of an INIT ACK with the OS value set to 0 SHOULD destroy the association discarding its TCB.

Number of Inbound Streams (MIS) : 16 bits (unsigned integer)

Defines the maximum number of streams the sender of this INIT ACK chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

Note: A receiver of an INIT ACK with the MIS value set to 0 SHOULD destroy the association discarding its TCB.

Initial TSN (I-TSN) : 32 bits (unsigned integer)

Defines the initial TSN that the INIT-ACK sender will use. The valid range is from 0 to 4294967295. This field MAY be set to the value of the Initiate Tag field.

| Fixed Parameters | Status |
|---|---|
| Initiate Tag | Mandatory |
| Advertised Receiver Window Credit | Mandatory |
| Number of Outbound Streams | Mandatory |
| Number of Inbound Streams | Mandatory |
| Initial TSN | Mandatory |

| Variable Parameters | Status | Type Value |
|---|---|---|
| State Cookie | Mandatory | 7 |
| IPv4 Address (Note 1) | Optional | 5 |
| IPv6 Address (Note 1) | Optional | 6 |
| Unrecognized Parameters | Optional | 8 |
| Reserved for ECN Capable (Note 2) | Optional | 32768 (0x8000) |
| Host Name Address (Note 3) | Optional | 11 |

Note 1: The INIT ACK chunks can contain any number of IP address parameters that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN capable field is reserved for future use of Explicit Congestion Notification.

Note 3: The INIT ACK chunks MUST NOT contain more than one Host Name address parameter. Moreover, the sender of the INIT ACK MUST NOT combine any other address types with the Host Name address in the INIT ACK. The receiver of the INIT ACK MUST ignore any other address types if the Host Name address parameter is present.

IMPLEMENTATION NOTE: An implementation MUST be prepared to receive a INIT ACK that is quite large (more than 1500 bytes) due to the variable size of the state cookie AND the variable address list. For example if a responder to the INIT has 1000 IPv4 addresses it wishes to send, it would need at least 8,000 bytes to encode this in the INIT ACK.

In combination with the Source Port carried in the SCTP common header, each IP Address parameter in the INIT ACK indicates to the receiver of the INIT ACK a valid transport address supported by the sender of the INIT ACK for the lifetime of the association being initiated.

If the INIT ACK contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT ACK and any additional address(es) provided within the INIT ACK may be used as destinations by the receiver of the INIT-ACK. If the INIT ACK does not contain any IP Address parameters, the receiver of the INIT-ACK MUST use the source address associated with the received IP datagram as its sole destination address for the association.

The State Cookie and Unrecognized Parameters use the Type-Length-Value format as defined in Section 3.2.1 and are described below. The other fields are defined the same as their counterparts in the INIT chunk.

3.3.3.1 Optional or Variable Length Parameters

State Cookie

Parameter Type Value: 7

Parameter Length: variable size, depending on Size of Cookie

Parameter Value:

This parameter value MUST contain all the necessary state and parameter information required for the sender of this INIT ACK to create the association, along with a Message Authentication Code (MAC). See Section 5.1.3 for details on State Cookie definition.

Unrecognized Parameters:

Parameter Type Value: 8

Parameter Length: Variable Size.

Parameter Value:

This parameter is returned to the originator of the INIT chunk when the INIT contains an unrecognized parameter which has a value that indicates that it should be reported to the sender. This parameter value field will contain unrecognized parameters copied from the INIT chunk complete with Parameter Type, Length and Value fields.

3.3.4 Selective Acknowledgement (SACK) (3):

This chunk is sent to the peer endpoint to acknowledge received DATA chunks and to inform the peer endpoint of gaps in the received subsequences of DATA chunks as represented by their TSNs.

The SACK MUST contain the Cumulative TSN Ack and Advertised Receiver Window Credit (a_rwnd) parameters.

By definition, the value of the Cumulative TSN Ack parameter is the last TSN received before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been received at the endpoint sending the SACK.  This parameter therefore acknowledges receipt of all TSNs less than or equal to its value.

The handling of a_rwnd by the receiver of the SACK is discussed in detail in Section 6.2.1.

The SACK also contains zero or more Gap Ack Blocks.  Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs.  By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 3    |Chunk  Flags   |      Chunk Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Cumulative TSN Ack                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Advertised Receiver Window Credit (a_rwnd)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Number of Gap Ack Blocks = N  |  Number of Duplicate TSNs = X |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
      |  Gap Ack Block #1 Start       |   Gap Ack Block #1 End
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /
/
      \                              ...
\
      /
/
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |   Gap Ack Block #N Start      |  Gap Ack Block #N End
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                       Duplicate TSN 1
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /
/
      \                              ...
\
      /
/
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                       Duplicate TSN X
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Chunk Flags: 8 bits

Set to all zeros on transmit and ignored on receipt.

Cumulative TSN Ack: 32 bits (unsigned integer)

This parameter contains the TSN of the last DATA chunk received in sequence before a gap.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This field indicates the updated receive buffer space in bytes of the sender of this SACK, see Section 6.2.1 for details.

Number of Gap Ack Blocks: 16 bits (unsigned integer)

Indicates the number of Gap Ack Blocks included in this SACK.

Number of Duplicate TSNs: 16 bit

This field contains the number of duplicate TSNs the endpoint has received.  Each duplicate TSN is listed following the Gap Ack Block list.

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly.

Gap Ack Block Start: 16 bits (unsigned integer)

Indicates the Start offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the first TSN in this Gap Ack Block that has been received.

Gap Ack Block End: 16 bits (unsigned integer)

Indicates the End offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the TSN of the last DATA chunk received in this Gap Ack Block.

For example, assume the receiver has the following DATA chunks newly arrived at the time when it decides to send a Selective ACK,

```
----------
| TSN=17 |
----------
|        | &lt;- still missing
----------
| TSN=15 |
----------
| TSN=14 |
----------
|        | &lt;- still missing
----------
| TSN=12 |
----------
| TSN=11 |
----------
| TSN=10 |
----------
```

then, the parameter part of the SACK MUST be constructed as follows (assuming the new a_rwnd is set to 4660 by the sender):

```
+--------------------------------+
|   Cumulative TSN Ack = 12      |
+--------------------------------+
|        a_rwnd = 4660           |
+----------------+---------------+
| num of block=2 | num of dup=0  |
+----------------+---------------+
```

```
                  |block #1 strrt=2 |block #1 end=3 |
```

Duplicate TSN: 32 bits (unsigned integer)

Indicates the number of times a TSN was received in duplicate since the last SACK was sent.  Every time a receiver gets a duplicate TSN (before sending the SACK) it adds it to the list of duplicates.  The duplicate count is re-initialized to zero after sending each SACK.

For example, if a receiver were to get the TSN 19 three times it would list 19 twice in the outbound SACK.  After sending the SACK if it received yet one more TSN 19 it would list 19 as a duplicate once in the next outgoing SACK.

3.3.5 Heartbeat Request (HEARTBEAT) (4):

An endpoint should send this chunk to its peer endpoint to probe the reachability of a particular destination transport address defined in the present association.

The parameter field contains the Heartbeat Information which is a variable length opaque data structure understood only by the sender.

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |   Type = 4    | Chunk  Flags  |      Heartbeat Length         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       \                                                               \
       /            Heartbeat Information TLV (Variable-Length)        /
       \                                                               \
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Heartbeat Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length

      Defined as a variable-length parameter using the format described
      in Section 3.2.1, i.e.:

```
      Variable Parameters                  Status     Type Value
      -------------------------------------------------------------
      Heartbeat Info                       Mandatory   1

       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |    Heartbeat Info Type=1      |         HB Info Length        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      /                  Sender-specific Heartbeat Info               /
      \                                                               \
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

      The Sender-specific Heartbeat Info field should normally include
      information about the sender's current time when this HEARTBEAT
      chunk is sent and the destination transport address to which this
      HEARTBEAT is sent (see Section 8.3).

3.3.6 Heartbeat Acknowledgement (HEARTBEAT ACK) (5):

An endpoint should send this chunk to its peer endpoint as a response to a HEARTBEAT chunk (see Section 8.3).  A HEARTBEAT ACK is always sent to the source IP address of the IP datagram containing the HEARTBEAT chunk to which this ack is responding.

The parameter field contains a variable length opaque data structure.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |   Type = 5    | Chunk  Flags  |    Heartbeat Ack Length       |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      \                                                               \
      /            Heartbeat Information TLV (Variable-Length)        /
      \                                                               \
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Heartbeat Ack Length:  16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length

This field MUST contain the Heartbeat Information parameter of the Heartbeat Request to which this Heartbeat Acknowledgement is responding.

| Variable Parameters | Status | Type Value |
| --- | --- | --- |
| Heartbeat Info | Mandatory | 1 |

3.3.7 Abort Association (ABORT) (6):

The ABORT chunk is sent to the peer of an association to close the association.  The ABORT chunk may contain Cause Parameters to inform the receiver the reason of the abort.  DATA chunks MUST NOT be bundled with ABORT.  Control chunks (except for INIT, INIT ACK and SHUTDOWN COMPLETE) MAY be bundled with an ABORT but they MUST be placed before the ABORT in the SCTP packet, or they will be ignored by the receiver.

If an endpoint receives an ABORT with a format error or for an association that doesn't exist, it MUST silently discard it. Moreover, under any circumstances, an endpoint that receives an ABORT MUST NOT respond to that ABORT by sending an ABORT of its own.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 6    |Reserved     |T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                   zero or more Error Causes                   /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Reserved:  7 bits

Set to 0 on transmit and ignored on receipt.

T bit:  1 bit

The T bit is set to 0 if the sender had a TCB that it destroyed. If the sender did not have a TCB it should set this bit to 1.

Note: Special rules apply to this chunk for verification, please see
Section 8.5.1 for details.

Length:  16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header
and all the Error Cause fields present.

See Section 3.3.10 for Error Cause definitions.

3.3.8 Shutdown Association (SHUTDOWN) (7):

An endpoint in an association MUST use this chunk to initiate a
graceful close of the association with its peer.  This chunk has the
following format.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Type = 7    | Chunk  Flags  |      Length = 8               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Cumulative TSN Ack                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

Length:  16 bits (unsigned integer)

Indicates the length of the parameter.  Set to 8.

Cumulative TSN Ack: 32 bits (unsigned integer)

This parameter contains the TSN of the last chunk received in
sequence before any gaps.

Note:  Since the SHUTDOWN message does not contain Gap Ack Blocks,
it cannot be used to acknowledge TSNs received out of order.  In a
SACK, lack of Gap Ack Blocks that were previously included
indicates that the data receiver reneged on the associated DATA
chunks.  Since SHUTDOWN does not contain Gap Ack Blocks, the
receiver of the SHUTDOWN shouldn't interpret the lack of a Gap Ack
Block as a renege. (see Section 6.2 for information on reneging)

3.3.9 Shutdown Acknowledgement (SHUTDOWN ACK) (8):

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN chunk at the completion of the shutdown process, see Section 9.2 for details.

The SHUTDOWN ACK chunk has no parameters.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    |   Type = 8    |Chunk  Flags   |      Length = 4
|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Chunk Flags:  8 bits

Set to zero on transmit and ignored on receipt.

3.3.10 Operation Error (ERROR) (9):

An endpoint sends this chunk to its peer endpoint to notify it of certain error conditions.  It contains one or more error causes. An Operation Error is not considered fatal in and of itself, but may be used with an ABORT chunk to report a fatal condition.  It has the following parameters:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    |   Type = 9    | Chunk  Flags  |           Length
|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    \
\
    /                    one or more Error Causes
/
    \
\
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Chunk Flags:  8 bits

Set to zero on transmit and ignored on receipt.

Length:  16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and all the Error Cause fields present.

Error causes are defined as variable-length parameters using the format described in 3.2.1, i.e.:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |           Cause Code          |       Cause Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /                    Cause-specific Information
/
      \
\
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Cause Code: 16 bits (unsigned integer)

Defines the type of error conditions being reported.

| Cause Code Value | Cause Code |
|---|---|
| 1 | Invalid Stream Identifier |
| 2 | Missing Mandatory Parameter |
| 3 | Stale Cookie Error |
| 4 | Out of Resource |
| 5 | Unresolvable Address |
| 6 | Unrecognized Chunk Type |
| 7 | Invalid Mandatory Parameter |
| 8 | Unrecognized Parameters |
| 9 | No User Data |
| 10 | Cookie Received While Shutting Down |

Cause Length: 16 bits (unsigned integer)

Set to the size of the parameter in bytes, including the Cause Code, Cause Length, and Cause-Specific Information fields

Cause-specific Information: variable length

This field carries the details of the error condition.

Sections 3.3.10.1 - 3.3.10.10 define error causes for SCTP. Guidelines for the IETF to define new error cause values are discussed in Section 13.3.

### 3.3.10.1 Invalid Stream Identifier (1)

Cause of error
---------------
Invalid Stream Identifier:  Indicates endpoint received a DATA chunk
sent to a nonexistent stream.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=1              |      Cause Length=8
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |        Stream Identifier      |         (Reserved)
|
```

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Stream Identifier: 16 bits (unsigned integer)

Contains the Stream Identifier of the DATA chunk received in error.

Reserved: 16 bits

This field is reserved. It is set to all 0's on transmit and Ignored on receipt.

3.3.10.2 Missing Mandatory Parameter (2)

Cause of error
---------------
Missing Mandatory Parameter: Indicates that one or more mandatory TLV parameters are missing in a received INIT or INIT ACK.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=2              |      Cause Length=8+N*2
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |                   Number of missing params=N
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |   Missing Param Type #1       |   Missing Param Type #2
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |   Missing Param Type #N-1     |   Missing Param Type #N
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Number of Missing params: 32 bits (unsigned integer)

This field contains the number of parameters contained in the Cause-specific Information field.

Missing Param Type: 16 bits (unsigned integer)

Each field will contain the missing mandatory parameter number.

3.3.10.3 Stale Cookie Error (3)

Cause of error
--------------
Stale Cookie Error: Indicates the receipt of a valid State Cookie that has expired.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=3              |       Cause Length=8
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

```
      |                  Measure of Staleness (usec.)
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

   Measure of Staleness:  32 bits (unsigned integer)

      This field contains the difference, in microseconds, between the
      current time and the time the State Cookie expired.

      The sender of this error cause MAY choose to report how long
past
      expiration the State Cookie is by including a non-zero value in
      the Measure of Staleness field.  If the sender does not wish to
      provide this information it should set the Measure of Staleness
      field to the value of zero.

3.3.10.4 Out of Resource (4)

   Cause of error
   ---------------
   Out of Resource: Indicates that the sender is out of resource.
This
   is usually sent in combination with or within an ABORT.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=4              |      Cause Length=4
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

3.3.10.5 Unresolvable Address (5)

   Cause of error
   ---------------
   Unresolvable Address: Indicates that the sender is not able to
   resolve the specified address parameter (e.g., type of address is
not
   supported by the sender).  This is usually sent in combination with
   or within an ABORT.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=5              |      Cause Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /                  Unresolvable Address
/
      \
\
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

   Unresolvable Address:  variable length

      The unresolvable address field contains the complete Type,
Length
      and Value of the address parameter (or Host Name parameter) that
      contains the unresolvable address or host name.

3.3.10.6 Unrecognized Chunk Type (6)

Cause of error
---------------
Unrecognized Chunk Type: This error cause is returned to the originator of the chunk if the receiver does not understand the chunk and the upper bits of the 'Chunk Type' are set to 01 or 11.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=6              |      Cause Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /                  Unrecognized Chunk
/
      \
\
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Unrecognized Chunk: variable length

The Unrecognized Chunk field contains the unrecognized Chunk from the SCTP packet complete with Chunk Type, Chunk Flags and Chunk Length.

3.3.10.7 Invalid Mandatory Parameter (7)

Cause of error
---------------
Invalid Mandatory Parameter: This error cause is returned to the originator of an INIT or INIT ACK chunk when one of the mandatory parameters is set to a invalid value.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=7              |      Cause Length=4
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

3.3.10.8 Unrecognized Parameters (8)

Cause of error
---------------
Unrecognized Parameters: This error cause is returned to the originator of the INIT ACK chunk if the receiver does not recognize one or more Optional TLV parameters in the INIT ACK chunk.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      |     Cause Code=8              |      Cause Length
|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
      /                  Unrecognized Parameters
/
```

```
      \                                                               \
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Unrecognized Parameters: variable length

The Unrecognized Parameters field contains the unrecognized parameters copied from the INIT ACK chunk complete with TLV. This error cause is normally contained in an ERROR chunk bundled with the COOKIE ECHO chunk when responding to the INIT ACK, when the sender of the COOKIE ECHO chunk wishes to report unrecognized parameters.

3.3.10.9 No User Data (9)

Cause of error
---------------
No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |     Cause Code=9              |      Cause Length=8           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      /                  TSN value                                    /
      \                                                               \
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TSN value: 32 bits (+unsigned integer)

The TSN value field contains the TSN of the DATA chunk received with no user data field.

This cause code is normally returned in an ABORT chunk (see Section 6.2)

3.3.10.10 Cookie Received While Shutting Down (10)

Cause of error
---------------
Cookie Received While Shutting Down: A COOKIE ECHO was received While the endpoint was in SHUTDOWN-ACK-SENT state. This error is usually returned in an ERROR chunk bundled with the retransmitted SHUTDOWN ACK.

```
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |     Cause Code=10             |      Cause Length=4           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.11 Cookie Echo (COOKIE ECHO) (10):

This chunk is used only during the initialization of an
association.
It is sent by the initiator of an association to its peer to
complete
the initialization process.  This chunk MUST precede any DATA chunk
sent within the association, but MAY be bundled with one or more
DATA
chunks in the same packet.

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |   Type = 10   |Chunk  Flags   |         Length
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       /                     Cookie
/
       \
\
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Chunk Flags: 8 bit

Set to zero on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the 4 bytes of
the chunk header and the size of the Cookie.

Cookie: variable size

This field must contain the exact cookie received in the State
Cookie parameter from the previous INIT ACK.

An implementation SHOULD make the cookie as small as possible to
insure interoperability.

3.3.12 Cookie Acknowledgement (COOKIE ACK) (11):

This chunk is used only during the initialization of an
association.
It is used to acknowledge the receipt of a COOKIE ECHO chunk.  This
chunk MUST precede any DATA or SACK chunk sent within the
association, but MAY be bundled with one or more DATA chunks or
SACK
chunk in the same SCTP packet.

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
       |   Type = 11   |Chunk  Flags   |     Length = 4
|
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Chunk Flags: 8 bits

Set to zero on transmit and ignored on receipt.

3.3.13 Shutdown Complete (SHUTDOWN COMPLETE) (14):

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN ACK chunk at the completion of the shutdown process, see Section 9.2 for details.

The SHUTDOWN COMPLETE chunk has no parameters.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 14   |Reserved     |T|      Length = 4               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Reserved:  7 bits

Set to 0 on transmit and ignored on receipt.

T bit:  1 bit

The T bit is set to 0 if the sender had a TCB that it destroyed. If the sender did not have a TCB it should set this bit to 1.

Note: Special rules apply to this chunk for verification, please see Section 8.5.1 for details.

4. SCTP Association State Diagram

During the lifetime of an SCTP association, the SCTP endpoint's association progress from one state to another in response to various events.  The events that may potentially advance an association's state include:

o  SCTP user primitive calls, e.g., [ASSOCIATE], [SHUTDOWN], [ABORT],

o  Reception of INIT, COOKIE ECHO, ABORT, SHUTDOWN, etc., control chunks, or

o  Some timeout events.

The state diagram in the figures below illustrates state changes, together with the causing events and resulting actions.  Note that some of the error conditions are not shown in the state diagram. Full description of all special cases should be found in the text.

Note: Chunk names are given in all capital letters, while parameter names have the first letter capitalized, e.g., COOKIE ECHO chunk type

vs. State Cookie parameter.  If more than one event/message can occur
which causes a state transition it is labeled (A), (B) etc.

```
                       -----          -------- (frm any state)
                     /       \      /  rcv ABORT      [ABORT]
    rcv INIT        |         |    |   ----------  or ----------
    --------------- |         v    v   delete TCB     snd ABORT
    generate Cookie  \    +---------+                 delete TCB
    snd INIT ACK       ---|  CLOSED |
                          +---------+
                           /      \      [ASSOCIATE]
                          /        \     ---------------
                         |          |    create TCB
                         |          |    snd INIT
                         |          |    strt init timer
          rcv valid      |          |
        COOKIE  ECHO     |          v
    (1) ---------------- |      +------------+
        create TCB       |      | COOKIE-WAIT| (2)
        snd COOKIE ACK   |      +------------+
                         |          |
                         |          |    rcv INIT ACK
                         |          |    -----------------
                         |          |    snd COOKIE ECHO
                         |          |    stop init timer
                         |          |    strt cookie timer
                         |          v
                         |      +--------------+
                         |      | COOKIE-ECHOED| (3)
                         |      +--------------+
                         |          |
                         |          |    rcv COOKIE ACK
                         |          |    -----------------
                         |          |    stop cookie timer
                         v          v
                       +---------------+
                       |  ESTABLISHED  |
                       +---------------+

                      (from the ESTABLISHED state only)
                                    |
                                    |
                           /--------+--------\
       [SHUTDOWN]         /                   \
       -------------------|                   |
       check outstanding  |                   |
       DATA chunks        |                   |
                          v                   |
                     +---------+              |
                     |SHUTDOWN-|              | rcv SHUTDOWN/check
                     |PENDING  |              | outstanding DATA
                     +---------+              | chunks
                          |                   |------------------
     No more outstanding  |                   |
     ---------------------|                   |
     snd SHUTDOWN         |                   |
     strt shutdown timer  |                   |
                          v                   v
                     +---------+        +-----------+
                 (4) |SHUTDOWN-|        | SHUTDOWN- |  (5,6)
```

```
                     |SENT     |        | RECEIVED  |
                     +---------+        +-----------+
                          |  \                 |
    (A) rcv SHUTDOWN ACK  |   \                |
    ----------------------|    \               |
    stop shutdown timer   |     \rcv:SHUTDOWN  |
    send SHUTDOWN COMPLETE|      \  (B)        |
    delete TCB            |       \            |
                          |        \           | No more outstanding
                          |         \          |-----------------
                          |          \         | send SHUTDOWN ACK
    (B)rcv SHUTDOWN       |           \        | strt shutdown timer
    ----------------------|            \       |
    send SHUTDOWN ACK     |             \      |
    start shutdown timer  |              \     |
    move to SHUTDOWN-     |               \    |
    ACK-SENT              |                |   |
                          |                v   |
                          |             +-----------+
                          |             | SHUTDOWN- | (7)
                          |             | ACK-SENT  |
                          |             +----------+-
                          |                    | (C)rcv SHUTDOWN
COMPLETE
                          |                    |-----------------
                          |                    | stop shutdown timer
                          |                    | delete TCB
                          |                    |

                          |                    | (D)rcv SHUTDOWN ACK
                          |                    |--------------
                          |                    | stop shutdown timer
                          |                    | send SHUTDOWN COMPLETE
                          |                    | delete TCB
                          |                    |
                          \    +---------+    /
                           \--&gt;| CLOSED  |&lt;--/
                               +---------+
```

Figure 3: State Transition Diagram of SCTP

Notes:

1) If the State Cookie in the received COOKIE ECHO is invalid (i.e., failed to pass the integrity check), the receiver MUST silently discard the packet. Or, if the received State Cookie is expired (see Section 5.1.5), the receiver MUST send back an ERROR chunk. In either case, the receiver stays in the CLOSED state.

2) If the T1-init timer expires, the endpoint MUST retransmit INIT and re-start the T1-init timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to SCTP user.

3) If the T1-cookie timer expires, the endpoint MUST retransmit COOKIE ECHO and re-start the T1-cookie timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times.

After that, the endpoint MUST abort the initialization process
and
report the error to SCTP user.

4) In SHUTDOWN-SENT state the endpoint MUST acknowledge any
received
DATA chunks without delay.

5) In SHUTDOWN-RECEIVED state, the endpoint MUST NOT accept any new send request from its SCTP user.

6) In SHUTDOWN-RECEIVED state, the endpoint MUST transmit or retransmit data and leave this state when all data in queue is transmitted.

7) In SHUTDOWN-ACK-SENT state, the endpoint MUST NOT accept any new send request from its SCTP user.

The CLOSED state is used to indicate that an association is not created (i.e., doesn't exist).

5. Association Initialization

Before the first data transmission can take place from one SCTP endpoint ("A") to another SCTP endpoint ("Z"), the two endpoints
must
complete an initialization process in order to set up an SCTP association between them.

The SCTP user at an endpoint should use the ASSOCIATE primitive to initialize an SCTP association to another SCTP endpoint.

IMPLEMENTATION NOTE: From an SCTP-user's point of view, an association may be implicitly opened, without an ASSOCIATE
primitive
(see 10.1 B) being invoked, by the initiating endpoint's sending of the first user data to the destination endpoint. The initiating
SCTP
will assume default values for all mandatory and optional
parameters
for the INIT/INIT ACK.

Once the association is established, unidirectional streams are
open
for data transfer on both ends (see Section 5.1.1).

5.1 Normal Establishment of an Association

The initialization process consists of the following steps
(assuming
that SCTP endpoint "A" tries to set up an association with SCTP endpoint "Z" and "Z" accepts the new association):

A) "A" first sends an INIT chunk to "Z". In the INIT, "A" must provide its Verification Tag (Tag_A) in the Initiate Tag field. Tag_A SHOULD be a random number in the range of 1 to 4294967295 (see 5.3.1 for Tag value selection). After sending the INIT,
"A"
starts the T1-init timer and enters the COOKIE-WAIT state.

B) "Z" shall respond immediately with an INIT ACK chunk. The destination IP address of the INIT ACK MUST be set to the source IP address of the INIT to which this INIT ACK is responding. In the response, besides filling in other parameters, "Z" must set the Verification Tag field to Tag_A, and also provide its own Verification Tag (Tag_Z) in the Initiate Tag field.

Moreover, "Z" MUST generate and send along with the INIT ACK a State Cookie. See Section 5.1.3 for State Cookie generation.

Note: After sending out INIT ACK with the State Cookie parameter, "Z" MUST NOT allocate any resources, nor keep any states for the new association. Otherwise, "Z" will be vulnerable to resource attacks.

C) Upon reception of the INIT ACK from "Z", "A" shall stop the T1-init timer and leave COOKIE-WAIT state. "A" shall then send the State Cookie received in the INIT ACK chunk in a COOKIE ECHO chunk, start the T1-cookie timer, and enter the COOKIE-ECHOED state.

Note: The COOKIE ECHO chunk can be bundled with any pending outbound DATA chunks, but it MUST be the first chunk in the packet and until the COOKIE ACK is returned the sender MUST NOT send any other packets to the peer.

D) Upon reception of the COOKIE ECHO chunk, Endpoint "Z" will reply with a COOKIE ACK chunk after building a TCB and moving to the ESTABLISHED state. A COOKIE ACK chunk may be bundled with any pending DATA chunks (and/or SACK chunks), but the COOKIE ACK chunk MUST be the first chunk in the packet.

IMPLEMENTATION NOTE: An implementation may choose to send the Communication Up notification to the SCTP user upon reception of a valid COOKIE ECHO chunk.

E) Upon reception of the COOKIE ACK, endpoint "A" will move from the COOKIE-ECHOED state to the ESTABLISHED state, stopping the T1-cookie timer. It may also notify its ULP about the successful establishment of the association with a Communication Up notification (see Section 10).

An INIT or INIT ACK chunk MUST NOT be bundled with any other chunk. They MUST be the only chunks present in the SCTP packets that carry them.

An endpoint MUST send the INIT ACK to the IP address from which it received the INIT.

Note: T1-init timer and T1-cookie timer shall follow the same rules given in Section 6.3.

If an endpoint receives an INIT, INIT ACK, or COOKIE ECHO chunk but decides not to establish the new association due to missing mandatory parameters in the received INIT or INIT ACK, invalid parameter values, or lack of local resources, it MUST respond with an ABORT chunk. It SHOULD also specify the cause of abort, such as the type of the missing mandatory parameters, etc., by including the error cause parameters with the ABORT chunk. The Verification Tag field in the common header of the outbound SCTP packet containing the ABORT chunk MUST be set to the Initiate Tag value of the peer.

After the reception of the first DATA chunk in an association the endpoint MUST immediately respond with a SACK to acknowledge the DATA chunk. Subsequent acknowledgements should be done as described in Section 6.2.

When the TCB is created, each endpoint MUST set its internal Cumulative TSN Ack Point to the value of its transmitted Initial TSN minus one.

IMPLEMENTATION NOTE: The IP addresses and SCTP port are generally used as the key to find the TCB within an SCTP instance.

5.1.1 Handle Stream Parameters

In the INIT and INIT ACK chunks, the sender of the chunk shall indicate the number of outbound streams (OS) it wishes to have in the association, as well as the maximum inbound streams (MIS) it will accept from the other endpoint.

After receiving the stream configuration information from the other side, each endpoint shall perform the following check: If the peer's MIS is less than the endpoint's OS, meaning that the peer is incapable of supporting all the outbound streams the endpoint wants to configure, the endpoint MUST either use MIS outbound streams, or abort the association and report to its upper layer the resources shortage at its peer.

After the association is initialized, the valid outbound stream identifier range for either endpoint shall be 0 to min(local OS, remote MIS)-1.

5.1.2 Handle Address Parameters

During the association initialization, an endpoint shall use the following rules to discover and collect the destination transport address(es) of its peer.

A) If there are no address parameters present in the received INIT or INIT ACK chunk, the endpoint shall take the source IP address from which the chunk arrives and record it, in combination with the SCTP source port number, as the only destination transport address for this peer.

B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint shall resolve that host name to a list of IP address(es) and derive the transport address(es) of this peer by combining the resolved IP address(es) with the SCTP source port.

The endpoint MUST ignore any other IP address parameters if they are also present in the received INIT or INIT ACK chunk.

The time at which the receiver of an INIT resolves the host name has potential security implications to SCTP. If the receiver of an INIT resolves the host name upon the reception of the chunk, and the mechanism the receiver uses to resolve the host name involves potential long delay (e.g. DNS query), the receiver may open itself up to resource attacks for the period of time while it is waiting for the name resolution results before it can build the State Cookie and release local resources.

Therefore, in cases where the name translation involves potential long delay, the receiver of the INIT MUST postpone the name resolution till the reception of the COOKIE ECHO chunk from the peer. In such a case, the receiver of the INIT SHOULD build the State Cookie using the received Host Name (instead of destination transport addresses) and send the INIT ACK to the source IP address from which the INIT was received.

The receiver of an INIT ACK shall always immediately attempt to resolve the name upon the reception of the chunk.

The receiver of the INIT or INIT ACK MUST NOT send user data (piggy-backed or stand-alone) to its peer until the host name is successfully resolved.

If the name resolution is not successful, the endpoint MUST immediately send an ABORT with "Unresolvable Address" error cause to its peer. The ABORT shall be sent to the source IP address from which the last peer packet was received.

C) If there are only IPv4/IPv6 addresses present in the received INIT or INIT ACK chunk, the receiver shall derive and record all the transport address(es) from the received chunk AND the source IP address that sent the INIT or INIT ACK. The transport address(es) are derived by the combination of SCTP source port (from the common header) and the IP address parameter(s) carried in the INIT or INIT ACK chunk and the source IP address of the IP datagram. The receiver should use only these transport addresses as destination transport addresses when sending subsequent packets to its peer.

IMPLEMENTATION NOTE: In some cases (e.g., when the implementation doesn't control the source IP address that is used for transmitting), an endpoint might need to include in its INIT or

INIT ACK all possible IP addresses from which packets to the peer could be transmitted.

After all transport addresses are derived from the INIT or INIT ACK chunk using the above rules, the endpoint shall select one of the transport addresses as the initial primary path.

Note: The INIT-ACK MUST be sent to the source address of the INIT.

The sender of INIT may include a 'Supported Address Types' parameter in the INIT to indicate what types of address are acceptable.  When this parameter is present, the receiver of INIT (initiatee) MUST either use one of the address types indicated in the Supported Address Types parameter when responding to the INIT, or abort the association with an "Unresolvable Address" error cause if it is unwilling or incapable of using any of the address types indicated by its peer.

IMPLEMENTATION NOTE: In the case that the receiver of an INIT ACK fails to resolve the address parameter due to an unsupported type, it can abort the initiation process and then attempt a re-initiation by using a 'Supported Address Types' parameter in the new INIT to indicate what types of address it prefers.

5.1.3 Generating State Cookie

When sending an INIT ACK as a response to an INIT chunk, the sender of INIT ACK creates a State Cookie and sends it in the State Cookie parameter of the INIT ACK.  Inside this State Cookie, the sender should include a MAC (see [<A HREF="/rfcs/rfc2104.html">RFC2104</A>] for an example), a time stamp on when the State Cookie is created, and the lifespan of the State Cookie, along with all the information necessary for it to establish the association.

The following steps SHOULD be taken to generate the State Cookie:

1) Create an association TCB using information from both the received INIT and the outgoing INIT ACK chunk,

2) In the TCB, set the creation time to the current time of day, and the lifespan to the protocol parameter 'Valid.Cookie.Life',

3) From the TCB, identify and collect the minimal subset of information needed to re-create the TCB, and generate a MAC using this subset of information and a secret key (see [<A HREF="/rfcs/rfc2104.html">RFC2104</A>] for an example of generating a MAC), and

4) Generate the State Cookie by combining this subset of information

and the resultant MAC.

After sending the INIT ACK with the State Cookie parameter, the
sender SHOULD delete the TCB and any other local resource related
to
the new association, so as to prevent resource attacks.

The hashing method used to generate the MAC is strictly a private
matter for the receiver of the INIT chunk.  The use of a MAC is
mandatory to prevent denial of service attacks.  The secret key
SHOULD be random ([<A HREF="/rfcs/rfc1750.html">RFC1750</A>]
provides some information on randomness
guidelines); it SHOULD be changed reasonably frequently, and the
timestamp in the State Cookie MAY be used to determine which key
should be used to verify the MAC.

An implementation SHOULD make the cookie as small as possible to
insure interoperability.

5.1.4 State Cookie Processing

When an endpoint (in the COOKIE WAIT state) receives an INIT ACK
chunk with a State Cookie parameter, it MUST immediately send a
COOKIE ECHO chunk to its peer with the received State Cookie.  The
sender MAY also add any pending DATA chunks to the packet after the
COOKIE ECHO chunk.

The endpoint shall also start the T1-cookie timer after sending out
the COOKIE ECHO chunk.  If the timer expires, the endpoint shall
retransmit the COOKIE ECHO chunk and restart the T1-cookie timer.
This is repeated until either a COOKIE ACK is received or '
Max.Init.Retransmits' is reached causing the peer endpoint to be
marked unreachable (and thus the association enters the CLOSED
state).

5.1.5 State Cookie Authentication

When an endpoint receives a COOKIE ECHO chunk from another endpoint
with which it has no association, it shall take the following
actions:

1) Compute a MAC using the TCB data carried in the State Cookie and
the secret key (note the timestamp in the State Cookie MAY be
used
to determine which secret key to use).  Reference [<A
HREF="/rfcs/rfc2104.html">RFC2104</A>] can be
used as a guideline for generating the MAC,

2) Authenticate the State Cookie as one that it previously
generated
by comparing the computed MAC against the one carried in the
State
Cookie.  If this comparison fails, the SCTP packet, including
the
COOKIE ECHO and any DATA chunks, should be silently discarded,

3) Compare the creation timestamp in the State Cookie to the
current
local time.  If the elapsed time is longer than the lifespan
carried in the State Cookie, then the packet, including the
COOKIE

ECHO and any attached DATA chunks, SHOULD be discarded and the endpoint MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint,

4) If the State Cookie is valid, create an association to the sender of the COOKIE ECHO chunk with the information in the TCB data carried in the COOKIE ECHO, and enter the ESTABLISHED state,

5) Send a COOKIE ACK chunk to the peer acknowledging reception of the COOKIE ECHO. The COOKIE ACK MAY be bundled with an outbound DATA chunk or SACK chunk; however, the COOKIE ACK MUST be the first chunk in the SCTP packet.

6) Immediately acknowledge any DATA chunk bundled with the COOKIE ECHO with a SACK (subsequent DATA chunk acknowledgement should follow the rules defined in Section 6.2). As mentioned in step 5), if the SACK is bundled with the COOKIE ACK, the COOKIE ACK MUST appear first in the SCTP packet.

If a COOKIE ECHO is received from an endpoint with which the receiver of the COOKIE ECHO has an existing association, the procedures in Section 5.2 should be followed.

5.1.6 An Example of Normal Association Establishment

In the following example, "A" initiates the association and then sends a user message to "Z", then "Z" sends two user messages to "A" later (assuming no bundling or fragmentation occurs):

```
   Endpoint A                                          Endpoint Z
   {app sets association with Z}
   (build TCB)
   INIT [I-Tag=Tag_A
         &amp; other info]  --------\
   (Start T1-init timer)          \
   (Enter COOKIE-WAIT state)      \---&gt; (compose temp TCB and
Cookie_Z)

                                   /--- INIT ACK [Veri Tag=Tag_A,
                                  /              I-Tag=Tag_Z,
   (Cancel T1-init timer) &lt;------/               Cookie_Z, &amp;
other info]
                                        (destroy temp TCB)
   COOKIE ECHO [Cookie_Z] ------\
   (Start T1-init timer)         \
   (Enter COOKIE-ECHOED state)    \---&gt; (build TCB enter
ESTABLISHED
                                         state)

                                  /---- COOKIE-ACK
                                 /
   (Cancel T1-init timer, &lt;-----/
    Enter ESTABLISHED state)
   {app sends 1st user data; strm 0}
   DATA [TSN=initial TSN_A
```

```
        Strm=0,Seq=1 &amp; user data]--\
     (Start T3-rtx timer)            \
                                      \-&gt;
                                  /----- SACK [TSN Ack=init
                                              TSN_A,Block=0]
   (Cancel T3-rtx timer) &lt;------/

                                         ...
                                         {app sends 2 messages;strm 0}
                                   /---- DATA
                                  /        [TSN=init TSN_Z
                              &lt;--/          Strm=0,Seq=1 &amp; user
data 1]
   SACK [TSN Ack=init TSN_Z,      /---- DATA
         Block=0]     --------\  /        [TSN=init TSN_Z +1,
                               \/          Strm=0,Seq=2 &amp; user
data 2]
                         &lt;------/\
                                  \
                                   \------&gt;
```

Figure 4: INITiation Example

If the T1-init timer expires at "A" after the INIT or COOKIE ECHO chunks are sent, the same INIT or COOKIE ECHO chunk with the same Initiate Tag (i.e., Tag_A) or State Cookie shall be retransmitted and

the timer restarted. This shall be repeated Max.Init.Retransmits times before "A" considers "Z" unreachable and reports the failure to its upper layer (and thus the association enters the CLOSED state). When retransmitting the INIT, the endpoint MUST follow the rules defined in 6.3 to determine the proper timer value.

## 5.2 Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK

During the lifetime of an association (in one of the possible states), an endpoint may receive from its peer endpoint one of the setup chunks (INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK). The receiver shall treat such a setup chunk as a duplicate and process it as described in this section.

Note: An endpoint will not receive the chunk unless the chunk was sent to a SCTP transport address and is from a SCTP transport address associated with this endpoint. Therefore, the endpoint processes such a chunk as part of its current association.

The following scenarios can cause duplicated or unexpected chunks:

A) The peer has crashed without being detected, re-started itself and sent out a new INIT chunk trying to restore the association,

B) Both sides are trying to initialize the association at about the same time,

C) The chunk is from a stale packet that was used to establish the present association or a past association that is no longer in existence,

D) The chunk is a false packet generated by an attacker, or

E) The peer never received the COOKIE ACK and is retransmitting its COOKIE ECHO.

The rules in the following sections shall be applied in order to identify and correctly handle these cases.

5.2.1 INIT received in COOKIE-WAIT or COOKIE-ECHOED State (Item B)

This usually indicates an initialization collision, i.e., each endpoint is attempting, at about the same time, to establish an association with the other endpoint.

Upon receipt of an INIT in the COOKIE-WAIT or COOKIE-ECHOED state, an endpoint MUST respond with an INIT ACK using the same parameters it

sent in its original INIT chunk (including its Initiation Tag, unchanged). These original parameters are combined with those from the newly received INIT chunk. The endpoint shall also generate a State Cookie with the INIT ACK. The endpoint uses the parameters sent in its INIT to calculate the State Cookie.

After that, the endpoint MUST NOT change its state, the T1-init timer shall be left running and the corresponding TCB MUST NOT be destroyed. The normal procedures for handling State Cookies when a TCB exists will resolve the duplicate INITs to a single association.

For an endpoint that is in the COOKIE-ECHOED state it MUST populate its Tie-Tags with the Tag information of itself and its peer (see section 5.2.2 for a description of the Tie-Tags).

5.2.2 Unexpected INIT in States Other than CLOSED, COOKIE-ECHOED, COOKIE-WAIT and SHUTDOWN-ACK-SENT

Unless otherwise stated, upon reception of an unexpected INIT for this association, the endpoint shall generate an INIT ACK with a State Cookie. In the outbound INIT ACK the endpoint MUST copy its current Verification Tag and peer's Verification Tag into a reserved place within the state cookie. We shall refer to these locations as the Peer's-Tie-Tag and the Local-Tie-Tag. The outbound SCTP packet containing this INIT ACK MUST carry a Verification Tag value equal to the Initiation Tag found in the unexpected INIT. And the INIT ACK MUST contain a new Initiation Tag (randomly generated see Section 5.3.1). Other parameters for the endpoint SHOULD be copied from the existing parameters of the association (e.g. number of outbound streams) into the INIT ACK and cookie.

After sending out the INIT ACK, the endpoint shall take no further actions, i.e., the existing association, including its current state, and the corresponding TCB MUST NOT be changed.

Note: Only when a TCB exists and the association is not in a COOKIE-WAIT state are the Tie-Tags populated.  For a normal association INIT (i.e. the endpoint is in a COOKIE-WAIT state), the Tie-Tags MUST be set to 0 (indicating that no previous TCB existed).  The INIT ACK and State Cookie are populated as specified in section 5.2.1.

5.2.3 Unexpected INIT ACK

If an INIT ACK is received by an endpoint in any state other than the COOKIE-WAIT state, the endpoint should discard the INIT ACK chunk. An unexpected INIT ACK usually indicates the processing of an old or duplicated INIT chunk.

5.2.4 Handle a COOKIE ECHO when a TCB exists

When a COOKIE ECHO chunk is received by an endpoint in any state for an existing association (i.e., not in the CLOSED state) the following rules shall be applied:

1) Compute a MAC as described in Step 1 of Section 5.1.5,

2) Authenticate the State Cookie as described in Step 2 of Section 5.1.5 (this is case C or D above).

3) Compare the timestamp in the State Cookie to the current time. If the State Cookie is older than the lifespan carried in the State Cookie and the Verification Tags contained in the State Cookie do not match the current association's Verification Tags, the packet, including the COOKIE ECHO and any DATA chunks, should be discarded.  The endpoint also MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint (this is case C or D in section 5.2).

   If both Verification Tags in the State Cookie match the Verification Tags of the current association, consider the State Cookie valid (this is case E of section 5.2) even if the lifespan is exceeded.

4) If the State Cookie proves to be valid, unpack the TCB into a temporary TCB.

5) Refer to Table 2 to determine the correct action to be taken.

+------------+------------+---------------+--------------+-------------
-+

| Local Tag | Peer's Tag | Local-Tie-Tag | Peer's-Tie-Tag | Action/ Description |
|---|---|---|---|---|
| X | X | M | M | (A) |
| M | X | A | A | (B) |
| M | 0 | A | A | (B) |
| X | M | 0 | 0 | (C) |
| M | M | A | A | (D) |

Table 2: Handling of a COOKIE ECHO when a TCB exists

Legend:

X - Tag does not match the existing TCB
M - Tag matches the existing TCB.
0 - No Tie-Tag in Cookie (unknown).
A - All cases, i.e. M, X or 0.

Note: For any case not shown in Table 2, the cookie should be silently discarded.

Action

A) In this case, the peer may have restarted. When the endpoint recognizes this potential 'restart', the existing session is treated the same as if it received an ABORT followed by a new COOKIE ECHO with the following exceptions:

- Any SCTP DATA Chunks MAY be retained (this is an implementation specific option).

- A notification of RESTART SHOULD be sent to the ULP instead of a "COMMUNICATION LOST" notification.

All the congestion control parameters (e.g., cwnd, ssthresh) related to this peer MUST be reset to their initial values (see Section 6.2.1).

After this the endpoint shall enter the ESTABLISHED state.

If the endpoint is in the SHUTDOWN-ACK-SENT state and recognizes the peer has restarted (Action A), it MUST NOT setup a new association but instead resend the SHUTDOWN ACK and send an ERROR chunk with a "Cookie Received while Shutting Down" error cause to its peer.

B) In this case, both sides may be attempting to start an association at about the same time but the peer endpoint started its INIT after responding to the local endpoint's INIT. Thus it may have picked a new Verification Tag not being aware of the previous Tag it had sent this endpoint. The endpoint should stay in or enter the ESTABLISHED state but it MUST update its peer's Verification Tag from the State Cookie, stop any init or cookie timers that may running and send a COOKIE ACK.

C) In this case, the local endpoint's cookie has arrived late. Before it arrived, the local endpoint sent an INIT and received an INIT-ACK and finally sent a COOKIE ECHO with the peer's same tag but a new tag of its own. The cookie should be silently discarded. The endpoint SHOULD NOT change states and should leave any timers running.

D) When both local and remote tags match the endpoint should always enter the ESTABLISHED state, if it has not already done so. It should stop any init or cookie timers that may be running and send a COOKIE ACK.

Note: The "peer's Verification Tag" is the tag received in the Initiate Tag field of the INIT or INIT ACK chunk.

5.2.4.1 An Example of a Association Restart

In the following example, "A" initiates the association after a restart has occurred. Endpoint "Z" had no knowledge of the restart until the exchange (i.e. Heartbeats had not yet detected the failure of "A"). (assuming no bundling or fragmentation occurs):

```
Endpoint A                                          Endpoint Z
&lt;-------------- Association is established----------------------
&gt;
Tag=Tag_A                                             Tag=Tag_Z
&lt;-----------------------------------------------------------------
&gt;
{A crashes and restarts}
{app sets up a association with Z}
(build TCB)
INIT [I-Tag=Tag_A'
      &amp; other info]  --------\
(Start T1-init timer)         \
(Enter COOKIE-WAIT state)      \---&gt; (find a existing TCB
                                      compose temp TCB and Cookie_Z
```

```
                                      with Tie-Tags to previous
                                      association)
                                /--- INIT ACK [Veri Tag=Tag_A',
                               /               I-Tag=Tag_Z',
(Cancel T1-init timer) &lt;------/                  Cookie_Z[TieTags=
                                               Tag_A,Tag_Z
                                                &amp; other info]
                                     (destroy temp TCB,leave original
                                      in place)
COOKIE ECHO [Veri=Tag_Z',
             Cookie_Z
             Tie=Tag_A,
             Tag_Z]----------\
(Start T1-init timer)         \
(Enter COOKIE-ECHOED state)    \---&gt; (Find existing association,
                                      Tie-Tags match old tags,
                                      Tags do not match i.e.
                                      case X X M M above,
                                      Announce Restart to ULP
                                      and reset association).
                               /---- COOKIE-ACK
                              /
(Cancel T1-init timer, &lt;-----/
 Enter ESTABLISHED state)
{app sends 1st user data; strm 0}
DATA [TSN=initial TSN_A
     Strm=0,Seq=1 &amp; user data]--\
(Start T3-rtx timer)            \
                                 \-&gt;
                              /----- SACK [TSN Ack=init TSN_A,Block=0]
(Cancel T3-rtx timer) &lt;------/
```

Figure 5: A Restart Example

### 5.2.5 Handle Duplicate COOKIE-ACK.

At any state other than COOKIE-ECHOED, an endpoint should silently discard a received COOKIE ACK chunk.

### 5.2.6 Handle Stale COOKIE Error

Receipt of an ERROR chunk with a "Stale Cookie" error cause indicates one of a number of possible events:

A) That the association failed to completely setup before the State Cookie issued by the sender was processed.

B) An old State Cookie was processed after setup completed.

C) An old State Cookie is received from someone that the receiver is not interested in having an association with and the ABORT chunk was lost.

When processing an ERROR chunk with a "Stale Cookie" error cause an endpoint should first examine if an association is in the process of being setup, i.e. the association is in the COOKIE-ECHOED state. In all cases if the association is not in the COOKIE-ECHOED state, the

ERROR chunk should be silently discarded.

If the association is in the COOKIE-ECHOED state, the endpoint may elect one of the following three alternatives.

1) Send a new INIT chunk to the endpoint to generate a new State Cookie and re-attempt the setup procedure.

2) Discard the TCB and report to the upper layer the inability to setup the association.

3) Send a new INIT chunk to the endpoint, adding a Cookie Preservative parameter requesting an extension to the lifetime of the State Cookie. When calculating the time extension, an implementation SHOULD use the RTT information measured based on the previous COOKIE ECHO / ERROR exchange, and should add no more than 1 second beyond the measured RTT, due to long State Cookie lifetimes making the endpoint more subject to a replay attack.

## 5.3 Other Initialization Issues

### 5.3.1 Selection of Tag Value

Initiate Tag values should be selected from the range of 1 to 2**32 - 1. It is very important that the Initiate Tag value be randomized to help protect against "man in the middle" and "sequence number" attacks. The methods described in [<A HREF="/rfcs/rfc1750.html">RFC1750</A>] can be used for the Initiate Tag randomization. Careful selection of Initiate Tags is also necessary to prevent old duplicate packets from previous associations being mistakenly processed as belonging to the current association.

Moreover, the Verification Tag value used by either endpoint in a given association MUST NOT change during the lifetime of an association. A new Verification Tag value MUST be used each time the endpoint tears-down and then re-establishes an association to the same peer.

## 6. User Data Transfer

Data transmission MUST only happen in the ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED states. The only exception to this is that DATA chunks are allowed to be bundled with an outbound COOKIE ECHO chunk when in COOKIE-WAIT state.

DATA chunks MUST only be received according to the rules below in ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT. A DATA chunk received in CLOSED is out of the blue and SHOULD be handled per 8.4. A DATA chunk received in any other state SHOULD be discarded.

A SACK MUST be processed in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED. An incoming SACK MAY be processed in COOKIE-

ECHOED. A SACK in the CLOSED state is out of the blue and SHOULD be processed according to the rules in 8.4. A SACK chunk received in any other state SHOULD be discarded.

A SCTP receiver MUST be able to receive a minimum of 1500 bytes in one SCTP packet. This means that a SCTP endpoint MUST NOT indicate less than 1500 bytes in its Initial a_rwnd sent in the INIT or INIT ACK.

For transmission efficiency, SCTP defines mechanisms for bundling of small user messages and fragmentation of large user messages. The following diagram depicts the flow of user messages through SCTP.

In this section the term "data sender" refers to the endpoint that transmits a DATA chunk and the term "data receiver" refers to the endpoint that receives a DATA chunk. A data receiver will transmit SACK chunks.

```
                 +--------------------------+
                 |      User Messages       |
                 +--------------------------+
       SCTP user        ^  |
      ==================|==|=======================================
                        |  v (1)
             +------------------+    +--------------------+
             | SCTP DATA Chunks |    |SCTP Control Chunks |
             +------------------+    +--------------------+
                        ^  |              ^  |
                        |  v (2)          |  v (2)
                     +--------------------------+
                     |      SCTP packets        |
                     +--------------------------+
       SCTP                      ^  |
      ===========================|==|===========================
                                 |  v
             Connectionless Packet Transfer Service (e.g., IP)
```

Notes:

1) When converting user messages into DATA chunks, an endpoint will fragment user messages larger than the current association path MTU into multiple DATA chunks. The data receiver will normally reassemble the fragmented message from DATA chunks before delivery to the user (see Section 6.9 for details).

2) Multiple DATA and control chunks may be bundled by the sender into a single SCTP packet for transmission, as long as the final size of the packet does not exceed the current path MTU. The receiver will unbundle the packet back into the original chunks. Control chunks MUST come before DATA chunks in the packet.

Figure 6: Illustration of User Data Transfer

The fragmentation and bundling mechanisms, as detailed in Sections 6.9 and 6.10, are OPTIONAL to implement by the data sender, but they

MUST be implemented by the data receiver, i.e., an endpoint MUST properly receive and process bundled or fragmented data.

6.1 Transmission of DATA Chunks

This document is specified as if there is a single retransmission timer per destination transport address, but implementations MAY have a retransmission timer for each DATA chunk.

The following general rules MUST be applied by the data sender for transmission and/or retransmission of outbound DATA chunks:

A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer's rwnd indicates that the peer has no buffer space (i.e. rwnd is 0, see Section 6.2.1). However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK having been lost in transit from the data receiver to the data sender.

B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd or more bytes of data outstanding to that transport address.

C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks which are marked for retransmission (limited by the current cwnd).

D) Then, the sender can send out as many new DATA chunks as Rule A and Rule B above allow.

Multiple DATA chunks committed for transmission MAY be bundled in a single packet. Furthermore, DATA chunks being retransmitted MAY be bundled with new DATA chunks, as long as the resulting packet size does not exceed the path MTU. A ULP may request that no bundling is performed but this should only turn off any delays that a SCTP implementation may be using to increase bundling efficiency. It does not in itself stop all bundling from occurring (i.e. in case of congestion or retransmission).

Before an endpoint transmits a DATA chunk, if any received DATA chunks have not been acknowledged (e.g., due to delayed ack), the sender should create a SACK and bundle it with the outbound DATA chunk, as long as the size of the final SCTP packet does not exceed the current MTU. See Section 6.2.

IMPLEMENTATION NOTE: When the window is full (i.e., transmission is disallowed by Rule A and/or Rule B), the sender MAY still accept send requests from its upper layer, but MUST transmit no more DATA chunks

until some or all of the outstanding DATA chunks are acknowledged and
transmission is allowed by Rule A and Rule B again.

Whenever a transmission or retransmission is made to any address, if
the T3-rtx timer of that address is not currently running, the sender
MUST start that timer. If the timer for that address is already
running, the sender MUST restart the timer if the earliest (i.e.,
lowest TSN) outstanding DATA chunk sent to that address is being
retransmitted. Otherwise, the data sender MUST NOT restart the
timer.

When starting or restarting the T3-rtx timer, the timer value must be
adjusted according to the timer rules defined in Sections 6.3.2, and
6.3.3.

Note: The data sender SHOULD NOT use a TSN that is more than 2**31 -
1 above the beginning TSN of the current send window.

6.2 Acknowledgement on Reception of DATA Chunks

The SCTP endpoint MUST always acknowledge the reception of each valid
DATA chunk.

The guidelines on delayed acknowledgement algorithm specified in
Section 4.2 of [<A HREF="/rfcs/rfc2581.html">RFC2581</A>] SHOULD be followed. Specifically, an
acknowledgement SHOULD be generated for at least every second packet
(not every second DATA chunk) received, and SHOULD be generated
within 200 ms of the arrival of any unacknowledged DATA chunk. In
some situations it may be beneficial for an SCTP transmitter to be
more conservative than the algorithms detailed in this document
allow. However, an SCTP transmitter MUST NOT be more aggressive than
the following algorithms allow.

A SCTP receiver MUST NOT generate more than one SACK for every
incoming packet, other than to update the offered window as the
receiving application consumes new data.

IMPLEMENTATION NOTE: The maximum delay for generating an
acknowledgement may be configured by the SCTP administrator, either
statically or dynamically, in order to meet the specific timing
requirement of the protocol being carried.

An implementation MUST NOT allow the maximum delay to be configured
to be more than 500 ms. In other words an implementation MAY lower
this value below 500ms but MUST NOT raise it above 500ms.

Acknowledgements MUST be sent in SACK chunks unless shutdown was
requested by the ULP in which case an endpoint MAY send an
acknowledgement in the SHUTDOWN chunk. A SACK chunk can acknowledge
the reception of multiple DATA chunks. See Section 3.3.4 for SACK

chunk format. In particular, the SCTP endpoint MUST fill in the Cumulative TSN Ack field to indicate the latest sequential TSN (of a valid DATA chunk) it has received. Any received DATA chunks with TSN greater than the value in the Cumulative TSN Ack field SHOULD also be reported in the Gap Ack Block fields.

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint should use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order .

When a packet arrives with duplicate DATA chunk(s) and with no new DATA chunk(s), the endpoint MUST immediately send a SACK with no delay. If a packet arrives with duplicate DATA chunk(s) bundled with new DATA chunks, the endpoint MAY immediately send a SACK. Normally receipt of duplicate DATA chunks will occur when the original SACK chunk was lost and the peer's RTO has expired. The duplicate TSN number(s) SHOULD be reported in the SACK as duplicate.

When an endpoint receives a SACK, it MAY use the Duplicate TSN information to determine if SACK loss is occurring. Further use of this data is for future study.

The data receiver is responsible for maintaining its receive buffers. The data receiver SHOULD notify the data sender in a timely manner of changes in its ability to receive data. How an implementation manages its receive buffers is dependent on many factors (e.g., Operating System, memory management system, amount of memory, etc.). However, the data sender strategy defined in Section 6.2.1 is based on the assumption of receiver operation similar to the following:

A) At initialization of the association, the endpoint tells the peer how much receive buffer space it has allocated to the association in the INIT or INIT ACK. The endpoint sets a_rwnd to this value.

B) As DATA chunks are received and buffered, decrement a_rwnd by the number of bytes received and buffered. This is, in effect, closing rwnd at the data sender and restricting the amount of data it can transmit.

C) As DATA chunks are delivered to the ULP and released from the receive buffers, increment a_rwnd by the number of bytes delivered to the upper layer. This is, in effect, opening up rwnd on the data sender and allowing it to send more data. The

data receiver SHOULD NOT increment a_rwnd unless it has released bytes from its receive buffer. For example, if the receiver is holding fragmented DATA chunks in a reassembly queue, it should not increment a_rwnd.

D) When sending a SACK, the data receiver SHOULD place the current value of a_rwnd into the a_rwnd field. The data receiver SHOULD take into account that the data sender will not retransmit DATA chunks that are acked via the Cumulative TSN Ack (i.e., will drop from its retransmit queue).

Under certain circumstances, the data receiver may need to drop DATA chunks that it has received but hasn't released from its receive buffers (i.e., delivered to the ULP). These DATA chunks may have been acked in Gap Ack Blocks. For example, the data receiver may be holding data in its receive buffers while reassembling a fragmented user message from its peer when it runs out of receive buffer space. It may drop these DATA chunks even though it has acknowledged them in Gap Ack Blocks. If a data receiver drops DATA chunks, it MUST NOT include them in Gap Ack Blocks in subsequent SACKs until they are received again via retransmission. In addition, the endpoint should take into account the dropped data when calculating its a_rwnd.

An endpoint SHOULD NOT revoke a SACK and discard data. Only in extreme circumstance should an endpoint use this procedure (such as out of buffer space). The data receiver should take into account that dropping data that has been acked in Gap Ack Blocks can result in suboptimal retransmission strategies in the data sender and thus in suboptimal performance.

The following example illustrates the use of delayed acknowledgements:

```
   Endpoint A                                      Endpoint Z

   {App sends 3 messages; strm 0}
   DATA [TSN=7,Strm=0,Seq=3] ------------&gt; (ack delayed)
   (Start T3-rtx timer)

   DATA [TSN=8,Strm=0,Seq=4] ------------&gt; (send ack)
                               /------- SACK [TSN Ack=8,block=0]
   (cancel T3-rtx timer)  &lt;-----/

   DATA [TSN=9,Strm=0,Seq=5] ------------&gt; (ack delayed)
   (Start T3-rtx timer)
                                        ...
                                        {App sends 1 message; strm
1}
                                        (bundle SACK with DATA)
                                 /----- SACK [TSN Ack=9,block=0] \
                                /         DATA [TSN=6,Strm=1,Seq=2]
   (cancel T3-rtx timer)  &lt;------/        (Start T3-rtx timer)

   (ack delayed)
   (send ack)
   SACK [TSN Ack=6,block=0] -------------&gt; (cancel T3-rtx timer)
```

Figure 7: Delayed Acknowledgment Example

If an endpoint receives a DATA chunk with no user data (i.e., the

Length field is set to 16) it MUST send an ABORT with error cause set to "No User Data".

An endpoint SHOULD NOT send a DATA chunk with no user data part.

### 6.2.1 Processing a Received SACK

Each SACK an endpoint receives contains an a_rwnd value. This value represents the amount of buffer space the data receiver, at the time of transmitting the SACK, has left of its total receive buffer space (as specified in the INIT/INIT ACK). Using a_rwnd, Cumulative TSN Ack and Gap Ack Blocks, the data sender can develop a representation of the peer's receive buffer space.

One of the problems the data sender must take into account when processing a SACK is that a SACK can be received out of order. That is, a SACK sent by the data receiver can pass an earlier SACK and be received first by the data sender. If a SACK is received out of order, the data sender can develop an incorrect view of the peer's receive buffer space.

Since there is no explicit identifier that can be used to detect out-of-order SACKs, the data sender must use heuristics to determine if a SACK is new.

An endpoint SHOULD use the following rules to calculate the rwnd, using the a_rwnd value, the Cumulative TSN Ack and Gap Ack Blocks in a received SACK.

A) At the establishment of the association, the endpoint initializes the rwnd to the Advertised Receiver Window Credit (a_rwnd) the peer specified in the INIT or INIT ACK.

B) Any time a DATA chunk is transmitted (or retransmitted) to a peer, the endpoint subtracts the data size of the chunk from the rwnd of that peer.

C) Any time a DATA chunk is marked for retransmission (via either T3-rtx timer expiration (Section 6.3.3)or via fast retransmit (Section 7.2.4)), add the data size of those chunks to the rwnd.

   Note: If the implementation is maintaining a timer on each DATA chunk then only DATA chunks whose timer expired would be marked for retransmission.

D) Any time a SACK arrives, the endpoint performs the following:

   i) If Cumulative TSN Ack is less than the Cumulative TSN Ack Point, then drop the SACK. Since Cumulative TSN Ack is monotonically increasing, a SACK whose Cumulative TSN Ack is less than the Cumulative TSN Ack Point indicates an out-of-order SACK.

ii) Set rwnd equal to the newly received a_rwnd minus the number of bytes still outstanding after processing the Cumulative TSN Ack and the Gap Ack Blocks.

iii) If the SACK is missing a TSN that was previously acknowledged via a Gap Ack Block (e.g., the data receiver reneged on the data), then mark the corresponding DATA chunk as available for retransmit: Mark it as missing for fast retransmit as described in Section 7.2.4 and if no retransmit timer is running for the destination address to which the DATA chunk was originally transmitted, then T3-rtx is started for that destination address.

### 6.3 Management of Retransmission Timer

An SCTP endpoint uses a retransmission timer T3-rtx to ensure data delivery in the absence of any feedback from its peer. The duration of this timer is referred to as RTO (retransmission timeout).

When an endpoint's peer is multi-homed, the endpoint will calculate a separate RTO for each different destination transport address of its peer endpoint.

The computation and management of RTO in SCTP follows closely how TCP manages its retransmission timer. To compute the current RTO, an endpoint maintains two state variables per destination transport address: SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation).

#### 6.3.1 RTO Calculation

The rules governing the computation of SRTT, RTTVAR, and RTO are as follows:

C1) Until an RTT measurement has been made for a packet sent to the given destination transport address, set RTO to the protocol parameter 'RTO.Initial'.

C2) When the first RTT measurement R is made, set SRTT <- R, RTTVAR <- R/2, and RTO <- SRTT + 4 * RTTVAR.

C3) When a new RTT measurement R' is made, set

RTTVAR <- (1 - RTO.Beta) * RTTVAR + RTO.Beta * |SRTT - R'|
SRTT <- (1 - RTO.Alpha) * SRTT + RTO.Alpha * R'

Note: The value of SRTT used in the update to RTTVAR is its value before updating SRTT itself using the second assignment.

After the computation, update RTO <- SRTT + 4 * RTTVAR.

C4) When data is in flight and when allowed by rule C5 below, a new
RTT measurement MUST be made each round trip. Furthermore, new
RTT measurements SHOULD be made no more than once per round-
trip
for a given destination transport address. There are two
reasons
for this recommendation: First, it appears that measuring more
frequently often does not in practice yield any significant
benefit [ALLMAN99]; second, if measurements are made more
often,
then the values of RTO.Alpha and RTO.Beta in rule C3 above
should
be adjusted so that SRTT and RTTVAR still adjust to changes at
roughly the same rate (in terms of how many round trips it
takes

them to reflect new values) as they would if making only one
measurement per round-trip and using RTO.Alpha and RTO.Beta as
given in rule C3. However, the exact nature of these
adjustments
remains a research issue.

C5) Karn's algorithm: RTT measurements MUST NOT be made using
packets
that were retransmitted (and thus for which it is ambiguous
whether the reply was for the first instance of the packet or a
later instance).

C6) Whenever RTO is computed, if it is less than RTO.Min seconds
then
it is rounded up to RTO.Min seconds. The reason for this rule
is
that RTOs that do not have a high minimum value are susceptible
to unnecessary timeouts [ALLMAN99].

C7) A maximum value may be placed on RTO provided it is at least
RTO.max seconds.

There is no requirement for the clock granularity G used for
computing RTT measurements and the different state variables, other
than:

G1) Whenever RTTVAR is computed, if RTTVAR = 0, then adjust RTTVAR
<-
G.

Experience [ALLMAN99] has shown that finer clock granularities
(<=
100 msec) perform somewhat better than more coarse granularities.

6.3.2 Retransmission Timer Rules

The rules for managing the retransmission timer are as follows:

R1) Every time a DATA chunk is sent to any address (including a
retransmission), if the T3-rtx timer of that address is not

running, start it running so that it will expire after the RTO
of
that address. The RTO used here is that obtained after any
doubling due to previous T3-rtx timer expirations on the
corresponding destination address as discussed in rule E2
below.

R2) Whenever all outstanding data sent to an address have been
acknowledged, turn off the T3-rtx timer of that address.

R3) Whenever a SACK is received that acknowledges the DATA chunk
with
the earliest outstanding TSN for that address, restart T3-rtx
timer for that address with its current RTO (if there is still
outstanding data on that address).

R4) Whenever a SACK is received missing a TSN that was previously
acknowledged via a Gap Ack Block, start T3-rtx for the
destination address to which the DATA chunk was originally
transmitted if it is not already running.

The following example shows the use of various timer rules
(assuming
the receiver uses delayed acks).

```
   Endpoint A                                        Endpoint Z
   {App begins to send}
   Data [TSN=7,Strm=0,Seq=3] ------------&gt; (ack delayed)
   (Start T3-rtx timer)
                                          {App sends 1 message; strm
1}
                                          (bundle ack with data)
   DATA [TSN=8,Strm=0,Seq=4] ----\     /-- SACK [TSN Ack=7,Block=0]
                                  \   /      DATA [TSN=6,Strm=1,Seq=2]
                                   \ /     (Start T3-rtx timer)
                                    \
                                   / \
   (Re-start T3-rtx timer) &lt;------/   \--&gt; (ack delayed)
   (ack delayed)
   {send ack}
   SACK [TSN Ack=6,Block=0] --------------&gt; (Cancel T3-rtx timer)
                                          ..
                                          (send ack)
   (Cancel T3-rtx timer)  &lt;-------------- SACK [TSN Ack=8,Block=0]
```

Figure 8 - Timer Rule Examples

### 6.3.3 Handle T3-rtx Expiration

Whenever the retransmission timer T3-rtx expires for a destination
address, do the following:

E1) For the destination address for which the timer expires, adjust
its ssthresh with rules defined in Section 7.2.3 and set the
cwnd
<- MTU.

E2) For the destination address for which the timer expires, set
RTO
<- RTO * 2 ("back off the timer"). The maximum value
discussed

in rule C7 above (RTO.max) may be used to provide an upper bound
to this doubling operation.

E3) Determine how many of the earliest (i.e., lowest TSN) outstanding
DATA chunks for the address for which the T3-rtx has expired will
fit into a single packet, subject to the MTU constraint for the
path corresponding to the destination transport address to which
the retransmission is being sent (this may be different from the

address for which the timer expires [see Section 6.4]). Call
this value K. Bundle and retransmit those K DATA chunks in a
single packet to the destination endpoint.

E4) Start the retransmission timer T3-rtx on the destination address
to which the retransmission is sent, if rule R1 above indicates
to do so. The RTO to be used for starting T3-rtx should be the
one for the destination address to which the retransmission is
sent, which, when the receiver is multi-homed, may be different
from the destination address for which the timer expired (see
Section 6.4 below).

After retransmitting, once a new RTT measurement is obtained (which
can happen only when new data has been sent and acknowledged, per
rule C5, or for a measurement made from a HEARTBEAT [see Section
8.3]), the computation in rule C3 is performed, including the
computation of RTO, which may result in "collapsing" RTO back down
after it has been subject to doubling (rule E2).

Note: Any DATA chunks that were sent to the address for which the
T3-rtx timer expired but did not fit in one MTU (rule E3 above),
should be marked for retransmission and sent as soon as cwnd allows
(normally when a SACK arrives).

The final rule for managing the retransmission timer concerns
failover (see Section 6.4.1):

F1) Whenever an endpoint switches from the current destination
transport address to a different one, the current retransmission
timers are left running. As soon as the endpoint transmits a
packet containing DATA chunk(s) to the new transport address,
start the timer on that transport address, using the RTO value of
the destination address to which the data is being sent, if rule
R1 indicates to do so.

6.4 Multi-homed SCTP Endpoints

An SCTP endpoint is considered multi-homed if there are more than one
transport address that can be used as a destination address to reach
that endpoint.

Moreover, the ULP of an endpoint shall select one of the multiple destination addresses of a multi-homed peer endpoint as the primary path (see Sections 5.1.2 and 10.1 for details).

By default, an endpoint SHOULD always transmit to the primary path, unless the SCTP user explicitly specifies the destination transport address (and possibly source transport address) to use.

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

When a receiver of a duplicate DATA chunk sends a SACK to a multi-homed endpoint it MAY be beneficial to vary the destination address and not use the source address of the DATA chunk. The reason being that receiving a duplicate from a multi-homed endpoint might indicate that the return path (as specified in the source address of the DATA chunk) for the SACK is broken.

Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

Retransmissions do not affect the total outstanding data count. However, if the DATA chunk is retransmitted onto a different destination address, both the outstanding data counts on the new destination address and the old destination address to which the data chunk was last sent shall be adjusted accordingly.

### 6.4.1 Failover from Inactive Destination Address

Some of the transport addresses of a multi-homed SCTP endpoint may become inactive due to either the occurrence of certain error conditions (see Section 8.2) or adjustments from SCTP user.

When there is outbound data to send and the primary path becomes inactive (e.g., due to failures), or where the SCTP user explicitly requests to send data to an inactive destination transport address, before reporting an error to its ULP, the SCTP endpoint should try to send the data to an alternate active destination transport address if one exists.

When retransmitting data, if the endpoint is multi-homed, it should consider each source-destination address pair in its retransmission selection policy.  When retransmitting the endpoint should attempt to
pick the most divergent source-destination pair from the original source-destination pair to which the packet was transmitted.

Note: Rules for picking the most divergent source-destination pair are an implementation decision and is not specified within this document.

## 6.5 Stream Identifier and Stream Sequence Number

Every DATA chunk MUST carry a valid stream identifier.  If an endpoint receives a DATA chunk with an invalid stream identifier, it
shall acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10) and discard the DATA
chunk. The endpoint may bundle the ERROR chunk in the same packet as
the SACK as long as the ERROR follows the SACK.

The stream sequence number in all the streams shall start from 0 when
the association is established.  Also, when the stream sequence number reaches the value 65535 the next stream sequence number shall
be set to 0.

## 6.6 Ordered and Unordered Delivery

Within a stream, an endpoint MUST deliver DATA chunks received with the U flag set to 0 to the upper layer according to the order of their stream sequence number.  If DATA chunks arrive out of order of
their stream sequence number, the endpoint MUST hold the received DATA chunks from delivery to the ULP until they are re-ordered.

However, an SCTP endpoint can indicate that no ordered delivery is required for a particular DATA chunk transmitted within the stream by
setting the U flag of the DATA chunk to 1.

When an endpoint receives a DATA chunk with the U flag set to 1, it must bypass the ordering mechanism and immediately deliver the data to the upper layer (after re-assembly if the user data is fragmented
by the data sender).

This provides an effective way of transmitting "out-of-band" data in
a given stream.  Also, a stream can be used as an "unordered" stream
by simply setting the U flag to 1 in all DATA chunks sent through that stream.

IMPLEMENTATION NOTE: When sending an unordered DATA chunk, an implementation may choose to place the DATA chunk in an outbound packet that is at the head of the outbound transmission queue if possible.

The 'Stream Sequence Number' field in a DATA chunk with U flag set to 1 has no significance. The sender can fill it with arbitrary value, but the receiver MUST ignore the field.

Note: When transmitting ordered and unordered data, an endpoint does not increment its Stream Sequence Number when transmitting a DATA chunk with U flag set to 1.

## 6.7 Report Gaps in Received DATA TSNs

Upon the reception of a new DATA chunk, an endpoint shall examine the continuity of the TSNs received. If the endpoint detects a gap in the received DATA chunk sequence, it SHOULD send a SACK with Gap Ack Blocks immediately. The data receiver continues sending a SACK after receipt of each SCTP packet that doesn't fill the gap.

Based on the Gap Ack Block from the received SACK, the endpoint can calculate the missing DATA chunks and make decisions on whether to retransmit them (see Section 6.2.1 for details).

Multiple gaps can be reported in one single SACK (see Section 3.3.4).

When its peer is multi-homed, the SCTP endpoint SHOULD always try to send the SACK to the same destination address from which the last DATA chunk was received.

Upon the reception of a SACK, the endpoint MUST remove all DATA chunks which have been acknowledged by the SACK's Cumulative TSN Ack from its transmit queue. The endpoint MUST also treat all the DATA chunks with TSNs not included in the Gap Ack Blocks reported by the SACK as "missing". The number of "missing" reports for each outstanding DATA chunk MUST be recorded by the data sender in order to make retransmission decisions. See Section 7.2.4 for details.

The following example shows the use of SACK to report a gap.

```
      Endpoint A                                    Endpoint Z
      {App sends 3 messages; strm 0}
      DATA [TSN=6,Strm=0,Seq=2] ---------------&gt; (ack delayed)
      (Start T3-rtx timer)

      DATA [TSN=7,Strm=0,Seq=3] --------&gt; X (lost)

      DATA [TSN=8,Strm=0,Seq=4] ---------------&gt; (gap detected,
                                                  immediately send
ack)
                                     /----- SACK [TSN Ack=6,Block=1,
                                    /             Strt=2,End=2]
                             &lt;-----/
      (remove 6 from out-queue,
```

and mark 7 as "1" missing report)

Figure 9 - Reporting a Gap using SACK

The maximum number of Gap Ack Blocks that can be reported within a single SACK chunk is limited by the current path MTU. When a single SACK can not cover all the Gap Ack Blocks needed to be reported due to the MTU limitation, the endpoint MUST send only one SACK, reporting the Gap Ack Blocks from the lowest to highest TSNs, within the size limit set by the MTU, and leave the remaining highest TSN numbers unacknowledged.

## 6.8 Adler-32 Checksum Calculation

When sending an SCTP packet, the endpoint MUST strengthen the data integrity of the transmission by including the Adler-32 checksum value calculated on the packet, as described below.

After the packet is constructed (containing the SCTP common header and one or more control or DATA chunks), the transmitter shall:

1) Fill in the proper Verification Tag in the SCTP common header and initialize the checksum field to 0's.

2) Calculate the Adler-32 checksum of the whole packet, including the SCTP common header and all the chunks. Refer to appendix B for details of the Adler-32 algorithm. And,

3) Put the resultant value into the checksum field in the common header, and leave the rest of the bits unchanged.

When an SCTP packet is received, the receiver MUST first check the Adler-32 checksum:

1) Store the received Adler-32 checksum value aside,

2) Replace the 32 bits of the checksum field in the received SCTP packet with all '0's and calculate an Adler-32 checksum value of the whole received packet. And,

3) Verify that the calculated Adler-32 checksum is the same as the received Adler-32 checksum. If not, the receiver MUST treat the packet as an invalid SCTP packet.

The default procedure for handling invalid SCTP packets is to silently discard them.

## 6.9 Fragmentation and Reassembly

An endpoint MAY support fragmentation when sending DATA chunks, but MUST support reassembly when receiving DATA chunks. If an endpoint supports fragmentation, it MUST fragment a user message if the size of the user message to be sent causes the outbound SCTP packet size to exceed the current MTU. If an implementation does not support fragmentation of outbound user messages, the endpoint must return an error to its upper layer and not attempt to send the user message.

IMPLEMENTATION NOTE: In this error case, the Send primitive discussed in Section 10.1 would need to return an error to the upper layer.

If its peer is multi-homed, the endpoint shall choose a size no larger than the association Path MTU. The association Path MTU is the smallest Path MTU of all destination addresses.

Note: Once a message is fragmented it cannot be re-fragmented. Instead if the PMTU has been reduced, then IP fragmentation must be used. Please see Section 7.3 for details of PMTU discovery.

When determining when to fragment, the SCTP implementation MUST take into account the SCTP packet header as well as the DATA chunk header(s). The implementation MUST also take into account the space required for a SACK chunk if bundling a SACK chunk with the DATA chunk.

Fragmentation takes the following steps:

1) The data sender MUST break the user message into a series of DATA chunks such that each chunk plus SCTP overhead fits into an IP datagram smaller than or equal to the association Path MTU.

2) The transmitter MUST then assign, in sequence, a separate TSN to each of the DATA chunks in the series. The transmitter assigns the same SSN to each of the DATA chunks. If the user indicates

   that the user message is to be delivered using unordered delivery, then the U flag of each DATA chunk of the user message MUST be set to 1.

3) The transmitter MUST also set the B/E bits of the first DATA chunk in the series to '10', the B/E bits of the last DATA chunk in the series to '01', and the B/E bits of all other DATA chunks in the series to '00'.

An endpoint MUST recognize fragmented DATA chunks by examining the B/E bits in each of the received DATA chunks, and queue the fragmented DATA chunks for re-assembly. Once the user message is reassembled, SCTP shall pass the re-assembled user message to the specific stream for possible re-ordering and final dispatching.

Note: If the data receiver runs out of buffer space while still waiting for more fragments to complete the re-assembly of the message, it should dispatch part of its inbound message through a partial delivery API (see Section 10), freeing some of its receive buffer space so that the rest of the message may be received.

6.10 Bundling

An endpoint bundles chunks by simply including multiple chunks in one
outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less or equal to the current Path MTU.

If its peer endpoint is multi-homed, the sending endpoint shall choose a size no larger than the latest MTU of the current primary path.

When bundling control chunks with DATA chunks, an endpoint MUST place
control chunks first in the outbound SCTP packet. The transmitter MUST transmit DATA chunks within a SCTP packet in increasing order of
TSN.

Note: Since control chunks must be placed first in a packet and since DATA chunks must be transmitted before SHUTDOWN or SHUTDOWN ACK
chunks, DATA chunks cannot be bundled with SHUTDOWN or SHUTDOWN ACK chunks.

Partial chunks MUST NOT be placed in an SCTP packet.

An endpoint MUST process received chunks in their order in the packet. The receiver uses the chunk length field to determine the end
of a chunk and beginning of the next chunk taking account of the fact
that all chunks end on a 4 byte boundary. If the receiver detects a
partial chunk, it MUST drop the chunk.

An endpoint MUST NOT bundle INIT, INIT ACK or SHUTDOWN COMPLETE with
any other chunks.

## 7. Congestion control

Congestion control is one of the basic functions in SCTP. For some applications, it may be likely that adequate resources will be allocated to SCTP traffic to assure prompt delivery of time-critical
data - thus it would appear to be unlikely, during normal operations,
that transmissions encounter severe congestion conditions. However SCTP must operate under adverse operational conditions, which can develop upon partial network failures or unexpected traffic surges. In such situations SCTP must follow correct congestion control steps
to recover from congestion quickly in order to get data delivered as
soon as possible. In the absence of network congestion, these preventive congestion control algorithms should show no impact on the
protocol performance.

IMPLEMENTATION NOTE: As far as its specific performance requirements
are met, an implementation is always allowed to adopt a more

conservative congestion control algorithm than the one defined below.

The congestion control algorithms used by SCTP are based on [<A HREF="/rfcs/rfc2581.html">RFC2581</A>]. This section describes how the algorithms defined in <A HREF="/rfcs/rfc2581.html">RFC2581</A> are adapted for use in SCTP. We first list differences in protocol designs between TCP and SCTP, and then describe SCTP's congestion control scheme. The description will use the same terminology as in TCP congestion control whenever appropriate.

SCTP congestion control is always applied to the entire association, and not to individual streams.

## 7.1 SCTP Differences from TCP Congestion control

Gap Ack Blocks in the SCTP SACK carry the same semantic meaning as the TCP SACK. TCP considers the information carried in the SACK as advisory information only. SCTP considers the information carried in the Gap Ack Blocks in the SACK chunk as advisory. In SCTP, any DATA chunk that has been acknowledged by SACK, including DATA that arrived at the receiving end out of order, are not considered fully delivered until the Cumulative TSN Ack Point passes the TSN of the DATA chunk (i.e., the DATA chunk has been acknowledged by the Cumulative TSN Ack

field in the SACK). Consequently, the value of cwnd controls the amount of outstanding data, rather than (as in the case of non-SACK TCP) the upper bound between the highest acknowledged sequence number and the latest DATA chunk that can be sent within the congestion window. SCTP SACK leads to different implementations of fast-retransmit and fast-recovery than non-SACK TCP. As an example see [FALL96].

The biggest difference between SCTP and TCP, however, is multi-homing. SCTP is designed to establish robust communication associations between two endpoints each of which may be reachable by more than one transport address. Potentially different addresses may lead to different data paths between the two endpoints, thus ideally one may need a separate set of congestion control parameters for each of the paths. The treatment here of congestion control for multi-homed receivers is new with SCTP and may require refinement in the future. The current algorithms make the following assumptions:

- o The sender usually uses the same destination address until being instructed by the upper layer otherwise; however, SCTP may change to an alternate destination in the event an address is marked inactive (see Section 8.2). Also, SCTP may retransmit to a different transport address than the original transmission.

o The sender keeps a separate congestion control parameter set for each of the destination addresses it can send to (not each source-destination pair but for each destination). The parameters should decay if the address is not used for a long enough time period.

o For each of the destination addresses, an endpoint does slow-start upon the first transmission to that address.

Note: TCP guarantees in-sequence delivery of data to its upper-layer protocol within a single TCP session. This means that when TCP notices a gap in the received sequence number, it waits until the gap is filled before delivering the data that was received with sequence numbers higher than that of the missing data. On the other hand, SCTP can deliver data to its upper-layer protocol even if there is a gap in TSN if the Stream Sequence Numbers are in sequence for a particular stream (i.e., the missing DATA chunks are for a different stream) or if unordered delivery is indicated. Although this does not affect cwnd, it might affect rwnd calculation.

7.2 SCTP Slow-Start and Congestion Avoidance

The slow start and congestion avoidance algorithms MUST be used by an endpoint to control the amount of data being injected into the network. The congestion control in SCTP is employed in regard to the association, not to an individual stream. In some situations it may be beneficial for an SCTP sender to be more conservative than the algorithms allow; however, an SCTP sender MUST NOT be more aggressive than the following algorithms allow.

Like TCP, an SCTP endpoint uses the following three control variables to regulate its transmission rate.

o Receiver advertised window size (rwnd, in bytes), which is set by the receiver based on its available buffer space for incoming packets.

  Note: This variable is kept on the entire association.

o Congestion control window (cwnd, in bytes), which is adjusted by the sender based on observed network conditions.

  Note: This variable is maintained on a per-destination address basis.

o Slow-start threshold (ssthresh, in bytes), which is used by the sender to distinguish slow start and congestion avoidance phases.

Note: This variable is maintained on a per-destination address basis.

SCTP also requires one additional control variable, partial_bytes_acked, which is used during congestion avoidance phase to facilitate cwnd adjustment.

Unlike TCP, an SCTP sender MUST keep a set of these control variables cwnd, ssthresh and partial_bytes_acked for EACH destination address of its peer (when its peer is multi-homed). Only one rwnd is kept for the whole association (no matter if the peer is multi-homed or has a single address).

### 7.2.1 Slow-Start

Beginning data transmission into a network with unknown conditions or after a sufficiently long idle period requires SCTP to probe the network to determine the available capacity. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

- o The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be <= 2*MTU.

- o The initial cwnd after a retransmission timeout MUST be no more than 1*MTU.

- o The initial value of ssthresh MAY be arbitrarily high (for example, implementations MAY use the size of the receiver advertised window).

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address.

- o When cwnd is less than or equal to ssthresh an SCTP endpoint MUST use the slow start algorithm to increase cwnd (assuming the current congestion window is being fully utilized). If an incoming SACK advances the Cumulative TSN Ack Point, cwnd MUST be increased by at most the lesser of 1) the total size of the previously outstanding DATA chunk(s) acknowledged, and 2) the destination's path MTU. This protects against the ACK-Splitting attack outlined in [SAVAGE99].

In instances where its peer endpoint is multi-homed, if an endpoint receives a SACK that advances its Cumulative TSN Ack Point, then it should update its cwnd (or cwnds) apportioned to the destination addresses to which it transmitted the acknowledged data. However if the received SACK does not advance the Cumulative TSN Ack Point, the endpoint MUST NOT adjust the cwnd of any of the destination addresses.

Because an endpoint's cwnd is not tied to its Cumulative TSN Ack Point, as duplicate SACKs come in, even though they may not advance the Cumulative TSN Ack Point an endpoint can still use them to clock out new data. That is, the data newly acknowledged by the SACK diminishes the amount of data now in flight to less than cwnd; and so the current, unchanged value of cwnd now allows new data to be sent.
On the other hand, the increase of cwnd must be tied to the Cumulative TSN Ack Point advancement as specified above. Otherwise the duplicate SACKs will not only clock out new data, but also will adversely clock out more new data than what has just left the network, during a time of possible congestion.

o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to max(cwnd/2, 2*MTU) per RTO.

7.2.2 Congestion Avoidance

When cwnd is greater than ssthresh, cwnd should be incremented by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address.

In practice an implementation can achieve this goal in the following way:

o partial_bytes_acked is initialized to 0.

o Whenever cwnd is greater than ssthresh, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.

o When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), increase cwnd by MTU, and reset partial_bytes_acked to (partial_bytes_acked - cwnd).

o Same as in the slow start, when the sender does not transmit DATA on a given transport address, the cwnd of the transport address should be adjusted to max(cwnd / 2, 2*MTU) per RTO.

o When all of the data transmitted by the sender has been acknowledged by the receiver, partial_bytes_acked is initialized to 0.

7.2.3 Congestion Control

Upon detection of packet losses from SACK (see Section 7.2.4), An endpoint should do the following:

```
      ssthresh = max(cwnd/2, 2*MTU)
      cwnd = ssthresh
```

Basically, a packet loss causes cwnd to be cut in half.

When the T3-rtx timer expires on an address, SCTP should perform slow
start by:

```
      ssthresh = max(cwnd/2, 2*MTU)
      cwnd = 1*MTU
```

and assure that no more than one SCTP packet will be in flight for that address until the endpoint receives acknowledgement for successful delivery of data to that address.

### 7.2.4 Fast Retransmit on Gap Reports

In the absence of data loss, an endpoint performs delayed acknowledgement. However, whenever an endpoint notices a hole in the
arriving TSN sequence, it SHOULD start sending a SACK back every time
a packet arrives carrying data until the hole is filled.

Whenever an endpoint receives a SACK that indicates some TSN(s) missing, it SHOULD wait for 3 further miss indications (via subsequent SACK's) on the same TSN(s) before taking action with regard to Fast Retransmit.

When the TSN(s) is reported as missing in the fourth consecutive SACK, the data sender shall:

1) Mark the missing DATA chunk(s) for retransmission,

2) Adjust the ssthresh and cwnd of the destination address(es) to which the missing DATA chunks were last sent, according to the formula described in Section 7.2.3.

3) Determine how many of the earliest (i.e., lowest TSN) DATA chunks
   marked for retransmission will fit into a single packet, subject to constraint of the path MTU of the destination transport address
   to which the packet is being sent. Call this value K. Retransmit
   those K DATA chunks in a single packet.

4) Restart T3-rtx timer only if the last SACK acknowledged the lowest
   outstanding TSN number sent to that address, or the endpoint is retransmitting the first outstanding DATA chunk sent to that address.

Note: Before the above adjustments, if the received SACK also acknowledges new DATA chunks and advances the Cumulative TSN Ack Point, the cwnd adjustment rules defined in Sections 7.2.1 and 7.2.2
must be applied first.

A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increments for each consecutive SACK reporting the TSN hole.  After reaching 4 and starting the fast retransmit procedure, the counter resets to 0.

Because cwnd in SCTP indirectly bounds the number of outstanding TSN's, the effect of TCP fast-recovery is achieved automatically with no adjustment to the congestion control window size.

7.3 Path MTU Discovery

[<A HREF="/rfcs/rfc1191.html">RFC1191</A>] specifies "Path MTU Discovery", whereby an endpoint maintains an estimate of the maximum transmission unit (MTU) along a given Internet path and refrains from sending packets along that path which exceed the MTU, other than occasional attempts to probe for a change in the Path MTU (PMTU).  <A HREF="/rfcs/rfc1191.html">RFC 1191</A> is thorough in its discussion of the MTU discovery mechanism and strategies for determining the current end-to-end MTU setting as well as detecting changes in this value.  [<A HREF="/rfcs/rfc1981.html">RFC1981</A>] specifies the same mechanisms for IPv6.  An SCTP sender using IPv6 MUST use Path MTU Discovery unless all packets are less than the minimum IPv6 MTU [<A HREF="/rfcs/rfc2460.html">RFC2460</A>].

An endpoint SHOULD apply these techniques, and SHOULD do so on a per-destination-address basis.

There are 4 ways in which SCTP differs from the description in RFC 1191 of applying MTU discovery to TCP:

1) SCTP associations can span multiple addresses.  An endpoint MUST maintain separate MTU estimates for each destination address of its peer.

2) Elsewhere in this document, when the term "MTU" is discussed, it refers to the MTU associated with the destination address corresponding to the context of the discussion.

3) Unlike TCP, SCTP does not have a notion of "Maximum Segment Size". Accordingly, the MTU for each destination address SHOULD be initialized to a value no larger than the link MTU for the local interface to which packets for that remote destination address will be routed.

4) Since data transmission in SCTP is naturally structured in terms of TSNs rather than bytes (as is the case for TCP), the discussion in Section 6.5 of <A HREF="/rfcs/rfc1191.html">RFC 1191</A> applies: When retransmitting an IP datagram to a remote address for which the IP datagram appears too large for the path MTU to that address, the IP datagram SHOULD be retransmitted without the DF bit set, allowing it to possibly be fragmented.  Transmissions of new IP datagrams MUST have DF set.

5) The sender should track an association PMTU which will be the smallest PMTU discovered for all of the peer's destination addresses. When fragmenting messages into multiple parts this association PMTU should be used to calculate the size of each fragment. This will allow retransmissions to be seamlessly sent to an alternate address without encountering IP fragmentation.

Other than these differences, the discussion of TCP's use of MTU discovery in RFCs 1191 and 1981 applies to SCTP on a per-destination-address basis.

Note: For IPv6 destination addresses the DF bit does not exist, instead the IP datagram must be fragmented as described in [<A HREF="/rfcs/rfc2460.html">RFC2460</A>].

## 8. Fault Management

### 8.1 Endpoint Failure Detection

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (including retransmissions to all the destination transport addresses of the peer if it is multi-homed). If the value of this counter exceeds the limit indicated in the protocol parameter 'Association.Max.Retrans', the endpoint shall consider the peer endpoint unreachable and shall stop transmitting any more data to it (and thus the association enters the CLOSED state). In addition, the endpoint shall report the failure to the upper layer, and optionally report back all outstanding user data remaining in its outbound queue. The association is automatically closed when the peer endpoint becomes unreachable.

The counter shall be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK), or a HEARTBEAT-ACK is received from the peer endpoint.

### 8.2 Path Failure Detection

When its peer endpoint is multi-homed, an endpoint should keep a error counter for each of the destination transport addresses of the peer endpoint.

Each time the T3-rtx timer expires on any address, or when a HEARTBEAT sent to an idle address is not acknowledged within a RTO, the error counter of that destination address will be incremented. When the value in the error counter exceeds the protocol parameter 'Path.Max.Retrans' of that destination address, the endpoint should mark the destination transport address as inactive, and a notification SHOULD be sent to the upper layer.

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint shall clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent). When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement should be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter may choose not to clear the error counter if the last chunk sent was a retransmission.

Note: When configuring the SCTP endpoint, the user should avoid having the value of 'Association.Max.Retrans' larger than the summation of the 'Path.Max.Retrans' of all the destination addresses for the remote endpoint. Otherwise, all the destination addresses may become inactive while the endpoint still considers the peer endpoint reachable. When this condition occurs, how the SCTP chooses to function is implementation specific.

When the primary path is marked inactive (due to excessive retransmissions, for instance), the sender MAY automatically transmit new packets to an alternate destination address if one exists and is active. If more than one alternate address is active when the primary path is marked inactive only ONE transport address SHOULD be chosen and used as the new destination transport address.

8.3 Path Heartbeat

By default, an SCTP endpoint shall monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es).

A destination transport address is considered "idle" if no new chunk which can be used for updating path RTT (usually including first transmission DATA, INIT, COOKIE ECHO, HEARTBEAT etc.) and no HEARTBEAT has been sent to it within the current heartbeat period of that address. This applies to both active and inactive destination addresses.

The upper layer can optionally initiate the following functions:

A) Disable heartbeat on a specific destination transport address of a given association,

B) Change the HB.interval,

C) Re-enable heartbeat on a specific destination transport address of a given association, and,

D) Request an on-demand HEARTBEAT on a specific destination transport address of a given association.

The endpoint should increment the respective error counter of the destination transport address each time a HEARTBEAT is sent to that address and not acknowledged within one RTO.

When the value of this counter reaches the protocol parameter ' Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

The sender of the HEARTBEAT chunk should include in the Heartbeat Information field of the chunk the current time when the packet is sent out and the destination address to which the packet is sent.

IMPLEMENTATION NOTE: An alternative implementation of the heartbeat mechanism that can be used is to increment the error counter variable every time a HEARTBEAT is sent to a destination. Whenever a HEARTBEAT ACK arrives, the sender SHOULD clear the error counter of the destination that the HEARTBEAT was sent to. This in effect would clear the previously stroked error (and any other error counts as well).

The receiver of the HEARTBEAT should immediately respond with a HEARTBEAT ACK that contains the Heartbeat Information field copied from the received HEARTBEAT chunk.

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in section 8.1).

The receiver of the HEARTBEAT ACK should also perform an RTT measurement for that destination transport address using the time value carried in the HEARTBEAT ACK chunk.

On an idle destination address that is allowed to heartbeat, a HEARTBEAT chunk is RECOMMENDED to be sent once per RTO of that destination address plus the protocol parameter 'HB.interval' , with jittering of +/- 50%, and exponential back-off of the RTO if the previous HEARTBEAT is unanswered.

A primitive is provided for the SCTP user to change the HB.interval and turn on or off the heartbeat on a given destination address. The heartbeat interval set by the SCTP user is added to the RTO of that destination (including any exponential backoff). Only one heartbeat should be sent each time the heartbeat timer expires (if multiple destinations are idle). It is a implementation decision on how to choose which of the candidate idle destinations to heartbeat to (if more than one destination is idle).

Note: When tuning the heartbeat interval, there is a side effect
that
SHOULD be taken into account. When this value is increased, i.e.
the HEARTBEAT takes longer, the detection of lost ABORT messages
takes longer as well. If a peer endpoint ABORTs the association
for
any reason and the ABORT chunk is lost, the local endpoint will
only
discover the lost ABORT by sending a DATA chunk or HEARTBEAT chunk
(thus causing the peer to send another ABORT). This must be
considered when tuning the HEARTBEAT timer. If the HEARTBEAT is
disabled only sending DATA to the association will discover a lost
ABORT from the peer.

8.4 Handle "Out of the blue" Packets

An SCTP packet is called an "out of the blue" (OOTB) packet if it
is
correctly formed, i.e., passed the receiver's Adler-32 check (see
Section 6.8), but the receiver is not able to identify the
association to which this packet belongs.

The receiver of an OOTB packet MUST do the following:

1) If the OOTB packet is to or from a non-unicast address, silently
discard the packet. Otherwise,

2) If the OOTB packet contains an ABORT chunk, the receiver MUST
silently discard the OOTB packet and take no further action.
Otherwise,

3) If the packet contains an INIT chunk with a Verification Tag set
to '0', process it as described in Section 5.1. Otherwise,

4) If the packet contains a COOKIE ECHO in the first chunk, process
it as described in Section 5.1. Otherwise,

5) If the packet contains a SHUTDOWN ACK chunk, the receiver should
respond to the sender of the OOTB packet with a SHUTDOWN
COMPLETE.
When sending the SHUTDOWN COMPLETE, the receiver of the OOTB
packet must fill in the Verification Tag field of the outbound
packet with the Verification Tag received in the SHUTDOWN ACK
and
set the T-bit in the Chunk Flags to indicate that no TCB was
found. Otherwise,

6) If the packet contains a SHUTDOWN COMPLETE chunk, the receiver
should silently discard the packet and take no further action.
Otherwise,

7) If the packet contains a "Stale cookie" ERROR or a COOKIE ACK
the
SCTP Packet should be silently discarded. Otherwise,

8) The receiver should respond to the sender of the OOTB packet
with
an ABORT. When sending the ABORT, the receiver of the OOTB
packet
MUST fill in the Verification Tag field of the outbound packet

with the value found in the Verification Tag field of the OOTB packet and set the T-bit in the Chunk Flags to indicate that no TCB was found. After sending this ABORT, the receiver of the OOTB packet shall discard the OOTB packet and take no further action.

8.5 Verification Tag

The Verification Tag rules defined in this section apply when sending or receiving SCTP packets which do not contain an INIT, SHUTDOWN COMPLETE, COOKIE ECHO (see Section 5.1), ABORT or SHUTDOWN ACK chunk. The rules for sending and receiving SCTP packets containing one of these chunk types are discussed separately in Section 8.5.1.

When sending an SCTP packet, the endpoint MUST fill in the Verification Tag field of the outbound packet with the tag value in the Initiate Tag parameter of the INIT or INIT ACK received from its peer.

When receiving an SCTP packet, the endpoint MUST ensure that the value in the Verification Tag field of the received SCTP packet matches its own Tag. If the received Verification Tag value does not match the receiver's own tag value, the receiver shall silently discard the packet and shall not process it any further except for those cases listed in Section 8.5.1 below.

8.5.1 Exceptions in Verification Tag Rules

A) Rules for packet carrying INIT:

- The sender MUST set the Verification Tag of the packet to 0.

- When an endpoint receives an SCTP packet with the Verification Tag set to 0, it should verify that the packet contains only an INIT chunk. Otherwise, the receiver MUST silently discard the packet.

B) Rules for packet carrying ABORT:

- The endpoint shall always fill in the Verification Tag field of the outbound packet with the destination endpoint's tag value if it is known.

- If the ABORT is sent in response to an OOTB packet, the endpoint MUST follow the procedure described in Section 8.4.

- The receiver MUST accept the packet if the Verification Tag matches either its own tag, OR the tag of its peer. Otherwise, the receiver MUST silently discard the packet and take no further action.

C) Rules for packet carrying SHUTDOWN COMPLETE:

- When sending a SHUTDOWN COMPLETE, if the receiver of the SHUTDOWN ACK has a TCB then the destination endpoint's tag MUST be used. Only where no TCB exists should the sender use the Verification Tag from the SHUTDOWN ACK.

- The receiver of a SHUTDOWN COMPLETE shall accept the packet if the Verification Tag field of the packet matches its own tag OR it is set to its peer's tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action. An endpoint MUST ignore the SHUTDOWN COMPLETE if it is not in the SHUTDOWN-ACK-SENT state.

D) Rules for packet carrying a COOKIE ECHO

- When sending a COOKIE ECHO, the endpoint MUST use the value of the Initial Tag received in the INIT ACK.

- The receiver of a COOKIE ECHO follows the procedures in Section 5.

E) Rules for packet carrying a SHUTDOWN ACK

- If the receiver is in COOKIE-ECHOED or COOKIE-WAIT state the procedures in section 8.4 SHOULD be followed, in other words it should be treated as an Out Of The Blue packet.

## 9. Termination of Association

An endpoint should terminate its association when it exits from service. An association can be terminated by either abort or shutdown. An abort of an association is abortive by definition in that any data pending on either end of the association is discarded and not delivered to the peer. A shutdown of an association is considered a graceful close where all data in queue by either endpoint is delivered to the respective peers. However, in the case of a shutdown, SCTP does not support a half-open state (like TCP) wherein one side may continue sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of sending or receiving the SHUTDOWN chunk.

### 9.1 Abort of an Association

When an endpoint decides to abort an existing association, it shall send an ABORT chunk to its peer endpoint. The sender MUST fill in the peer's Verification Tag in the outbound packet and MUST NOT bundle any DATA chunk with the ABORT.

An endpoint MUST NOT respond to any received packet that contains an
ABORT chunk (also see Section 8.4).

An endpoint receiving an ABORT shall apply the special Verification Tag check rules described in Section 8.5.1.

After checking the Verification Tag, the receiving endpoint shall remove the association from its record, and shall report the termination to its upper layer.

## 9.2 Shutdown of an Association

Using the SHUTDOWN primitive (see Section 10.1), the upper layer of an endpoint in an association can gracefully close the association. This will allow all outstanding DATA chunks from the peer of the shutdown initiator to be delivered before the association terminates.

Upon receipt of the SHUTDOWN primitive from its upper layer, the endpoint enters SHUTDOWN-PENDING state and remains there until all outstanding data has been acknowledged by its peer. The endpoint

accepts no new data from its upper layer, but retransmits data to the
far end if necessary to fill gaps.

Once all its outstanding data has been acknowledged, the endpoint shall send a SHUTDOWN chunk to its peer including in the Cumulative TSN Ack field the last sequential TSN it has received from the peer.
It shall then start the T2-shutdown timer and enter the SHUTDOWN-SENT
state. If the timer expires, the endpoint must re-send the SHUTDOWN
with the updated last sequential TSN received from its peer.

The rules in Section 6.3 MUST be followed to determine the proper timer value for T2-shutdown. To indicate any gaps in TSN, the endpoint may also bundle a SACK with the SHUTDOWN chunk in the same SCTP packet.

An endpoint should limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded the endpoint should destroy the TCB and
MUST report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state). The reception of any
packet from its peer (i.e. as the peer sends all of its queued DATA chunks) should clear the endpoint's retransmission count and restart
the T2-Shutdown timer, giving its peer ample opportunity to transmit
all of its queued DATA chunks that have not yet been sent.

Upon the reception of the SHUTDOWN, the peer endpoint shall

- enter the SHUTDOWN-RECEIVED state,

- stop accepting new data from its SCTP user

- verify, by checking the Cumulative TSN Ack field of the chunk, that all its outstanding DATA chunks have been received by the SHUTDOWN sender.

Once an endpoint as reached the SHUTDOWN-RECEIVED state it MUST NOT send a SHUTDOWN in response to a ULP request, and should discard subsequent SHUTDOWN chunks.

If there are still outstanding DATA chunks left, the SHUTDOWN receiver shall continue to follow normal data transmission procedures defined in Section 6 until all outstanding DATA chunks are acknowledged; however, the SHUTDOWN receiver MUST NOT accept new data from its SCTP user.

While in SHUTDOWN-SENT state, the SHUTDOWN sender MUST immediately respond to each received packet containing one or more DATA chunk(s) with a SACK, a SHUTDOWN chunk, and restart the T2-shutdown timer. If

it has no more outstanding DATA chunks, the SHUTDOWN receiver shall send a SHUTDOWN ACK and start a T2-shutdown timer of its own, entering the SHUTDOWN-ACK-SENT state. If the timer expires, the endpoint must re-send the SHUTDOWN ACK.

The sender of the SHUTDOWN ACK should limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded the endpoint should destroy the TCB and may report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

Upon the receipt of the SHUTDOWN ACK, the SHUTDOWN sender shall stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer, and remove all record of the association.

Upon reception of the SHUTDOWN COMPLETE chunk the endpoint will verify that it is in SHUTDOWN-ACK-SENT state, if it is not the chunk should be discarded. If the endpoint is in the SHUTDOWN-ACK-SENT state the endpoint should stop the T2-shutdown timer and remove all knowledge of the association (and thus the association enters the CLOSED state).

An endpoint SHOULD assure that all its outstanding DATA chunks have been acknowledged before initiating the shutdown procedure.

An endpoint should reject any new data request from its upper layer if it is in SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, or SHUTDOWN-ACK-SENT state.

If an endpoint is in SHUTDOWN-ACK-SENT state and receives an INIT chunk (e.g., if the SHUTDOWN COMPLETE was lost) with source and destination transport addresses (either in the IP addresses or in the

INIT chunk) that belong to this association, it should discard the INIT chunk and retransmit the SHUTDOWN ACK chunk.

Note: Receipt of an INIT with the same source and destination IP addresses as used in transport addresses assigned to an endpoint but with a different port number indicates the initialization of a separate association.

The sender of the INIT or COOKIE ECHO should respond to the receipt of a SHUTDOWN-ACK with a stand-alone SHUTDOWN COMPLETE in an SCTP packet with the Verification Tag field of its common header set to the same tag that was received in the SHUTDOWN ACK packet. This is considered an Out of the Blue packet as defined in Section 8.4. The sender of the INIT lets T1-init continue running and remains in the

COOKIE-WAIT or COOKIE-ECHOED state. Normal T1-init timer expiration will cause the INIT or COOKIE chunk to be retransmitted and thus start a new association.

If a SHUTDOWN is received in COOKIE WAIT or COOKIE ECHOED states the SHUTDOWN chunk SHOULD be silently discarded.

If an endpoint is in SHUTDOWN-SENT state and receives a SHUTDOWN chunk from its peer, the endpoint shall respond immediately with a SHUTDOWN ACK to its peer, and move into a SHUTDOWN-ACK-SENT state restarting its T2-shutdown timer.

If an endpoint is in the SHUTDOWN-ACK-SENT state and receives a SHUTDOWN ACK, it shall stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer, and remove all record of the association.

## 10. Interface with Upper Layer

The Upper Layer Protocols (ULP) shall request for services by passing primitives to SCTP and shall receive notifications from SCTP for various events.

The primitives and notifications described in this section should be used as a guideline for implementing SCTP. The following functional description of ULP interface primitives is shown for illustrative purposes. Different SCTP implementations may have different ULP interfaces. However, all SCTPs must provide a certain minimum set of services to guarantee that all SCTP implementations can support the same protocol hierarchy.

### 10.1 ULP-to-SCTP

The following sections functionally characterize a ULP/SCTP interface. The notation used is similar to most procedure or function calls in high level languages.

The ULP primitives described below specify the basic functions the

SCTP must perform to support inter-process communication. Individual
implementations must define their own exact format, and may provide
combinations or subsets of the basic functions in single calls.

A) Initialize

Format: INITIALIZE ([local port], [local eligible address list]) ->
local SCTP instance name

This primitive allows SCTP to initialize its internal data structures
and allocate necessary resources for setting up its operation
environment. Once SCTP is initialized, ULP can communicate directly
with other endpoints without re-invoking this primitive.

SCTP will return a local SCTP instance name to the ULP.

Mandatory attributes:

None.

Optional attributes:

The following types of attributes may be passed along with the
primitive:

o local port - SCTP port number, if ULP wants it to be specified;

o local eligible address list - An address list that the local SCTP
endpoint should bind. By default, if an address list is not
included, all IP addresses assigned to the host should be used by
the local endpoint.

IMPLEMENTATION NOTE: If this optional attribute is supported by an
implementation, it will be the responsibility of the implementation
to enforce that the IP source address field of any SCTP packets sent
out by this endpoint contains one of the IP addresses indicated in
the local eligible address list.

B) Associate

Format: ASSOCIATE(local SCTP instance name, destination transport addr,
outbound stream count)
-> association id [,destination transport addr list] [,outbound stream
count]

This primitive allows the upper layer to initiate an association to a
specific peer endpoint.

The peer endpoint shall be specified by one of the transport
addresses which defines the endpoint (see Section 1.4). If the local

SCTP instance has not been initialized, the ASSOCIATE is considered an error.

An association id, which is a local handle to the SCTP association, will be returned on successful establishment of the association. If SCTP is not able to open an SCTP association with the peer endpoint, an error is returned.

Other association parameters may be returned, including the complete destination transport addresses of the peer as well as the outbound stream count of the local endpoint. One of the transport address from the returned destination addresses will be selected by the local endpoint as default primary path for sending SCTP packets to this peer. The returned "destination transport addr list" can be used by the ULP to change the default primary path or to force sending a packet to a specific transport address.

IMPLEMENTATION NOTE: If ASSOCIATE primitive is implemented as a blocking function call, the ASSOCIATE primitive can return association parameters in addition to the association id upon successful establishment. If ASSOCIATE primitive is implemented as a non-blocking call, only the association id shall be returned and association parameters shall be passed using the COMMUNICATION UP notification.

Mandatory attributes:

o local SCTP instance name - obtained from the INITIALIZE operation.

o destination transport addr - specified as one of the transport addresses of the peer endpoint with which the association is to be established.

o outbound stream count - the number of outbound streams the ULP would like to open towards this peer endpoint.

Optional attributes:

None.

C) Shutdown

Format: SHUTDOWN(association id)
-> result

Gracefully closes an association. Any locally queued user data will be delivered to the peer. The association will be terminated only after the peer acknowledges all the SCTP packets sent. A success code will be returned on successful termination of the association. If attempting to terminate the association results in a failure, an error code shall be returned.

Mandatory attributes:

o association id - local handle to the SCTP association

Optional attributes:

None.

D) Abort

Format: ABORT(association id [, cause code])
-> result

Ungracefully closes an association. Any locally queued user data will be discarded and an ABORT chunk is sent to the peer. A success code will be returned on successful abortion of the association. If attempting to abort the association results in a failure, an error code shall be returned.

Mandatory attributes:

o association id - local handle to the SCTP association

Optional attributes:

o cause code - reason of the abort to be passed to the peer.

None.

E) Send

Format: SEND(association id, buffer address, byte count [,context]
        [,stream id] [,life time] [,destination transport address]
        [,unorder flag] [,no-bundle flag] [,payload protocol-id] )
-> result

This is the main method to send user data via SCTP.

Mandatory attributes:

o association id - local handle to the SCTP association

o buffer address - the location where the user message to be transmitted is stored;

o byte count - The size of the user data in number of bytes;

Optional attributes:

o context - an optional 32 bit integer that will be carried in the sending failure notification to the ULP if the transportation of this User Message fails.

o stream id - to indicate which stream to send the data on. If not specified, stream 0 will be used.

o life time - specifies the life time of the user data. The user data will not be sent by SCTP after the life time expires. This parameter can be used to avoid efforts to transmit stale user messages. SCTP notifies the ULP if the data cannot be initiated to transport (i.e. sent to the destination via SCTP's send primitive) within the life time variable. However, the user data will be transmitted if SCTP has attempted to transmit a chunk before the life time expired.

IMPLEMENTATION NOTE: In order to better support the data lifetime option, the transmitter may hold back the assigning of the TSN number to an outbound DATA chunk to the last moment. And, for implementation simplicity, once a TSN number has been assigned the sender should consider the send of this DATA chunk as committed, overriding any lifetime option attached to the DATA chunk.

o destination transport address - specified as one of the destination transport addresses of the peer endpoint to which this packet should be sent. Whenever possible, SCTP should use this destination transport address for sending the packets, instead of the current primary path.

o unorder flag - this flag, if present, indicates that the user would like the data delivered in an unordered fashion to the peer (i.e., the U flag is set to 1 on all DATA chunks carrying this message).

o no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP MAY still bundle even when this flag is present, when faced with network congestion.

o payload protocol-id - A 32 bit unsigned integer that is to be passed to the peer indicating the type of payload protocol data being transmitted. This value is passed as opaque data by SCTP.

F) Set Primary

Format: SETPRIMARY(association id, destination transport address,
                   [source transport address] )
-> result

Instructs the local SCTP to use the specified destination transport address as primary path for sending packets.

The result of attempting this operation shall be returned. If the specified destination transport address is not present in the "destination transport address list" returned earlier in an associate command or communication up notification, an error shall be returned.

Mandatory attributes:

o association id - local handle to the SCTP association o  destination transport address - specified as one of the transport addresses of the peer endpoint, which should be used as primary address for sending packets.  This overrides the current primary address information maintained by the local SCTP endpoint.

Optional attributes:

o  source transport address - optionally, some implementations may allow you to set the default source address placed in all outgoing IP datagrams.

G) Receive

Format: RECEIVE(association id, buffer address, buffer size [,stream id])
-> byte count [,transport address] [,stream id] [,stream sequence number] [,partial flag] [,delivery number] [,payload protocol-id]

This primitive shall read the first user message in the SCTP in-queue into the buffer specified by ULP, if there is one available.  The size of the message read, in bytes, will be returned.  It may, depending on the specific implementation, also return other information such as the sender's address, the stream id on which it is received, whether there are more messages available for retrieval, etc.  For ordered messages, their stream sequence number may also be returned.

Depending upon the implementation, if this primitive is invoked when no message is available the implementation should return an indication of this condition or should block the invoking process until data does become available.

Mandatory attributes:

o  association id - local handle to the SCTP association

o  buffer address - the memory location indicated by the ULP to store the received message.

o  buffer size - the maximum size of data to be received, in bytes.

Optional attributes:

o  stream id - to indicate which stream to receive the data on.

o  stream sequence number - the stream sequence number assigned by the sending SCTP peer.

o  partial flag - if this returned flag is set to 1, then this Receive contains  a partial delivery of the whole message.  When this flag is set, the stream id and stream sequence number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

o payload protocol-id - A 32 bit unsigned integer that is received from the peer indicating the type of payload protocol of the received data. This value is passed as opaque data by SCTP.

H) Status

Format: STATUS(association id)
-> status data

This primitive should return a data block containing the following information:
association connection state,
destination transport address list,
destination transport address reachability states,
current receiver window size,
current congestion window sizes,
number of  unacknowledged DATA chunks,
number of DATA chunks pending receipt,
primary path,
most recent SRTT on primary path,
RTO on primary path,
SRTT and RTO on other destination addresses, etc.

Mandatory attributes:

o association id - local handle to the SCTP association

Optional attributes:

None.

I) Change Heartbeat

Format: CHANGEHEARTBEAT(association id, destination transport address,
new state [,interval])
-> result

Instructs the local endpoint to enable or disable heartbeat on the specified destination transport address.

The result of attempting this operation shall be returned.

Note: Even when enabled, heartbeat will not take place if the destination transport address is not idle.

Mandatory attributes:

o association id - local handle to the SCTP association

o destination transport address - specified as one of the transport addresses of the peer endpoint.

o new state - the new state of heartbeat for this destination transport address (either enabled or disabled).

Optional attributes:

o interval - if present, indicates the frequency of the heartbeat if this is to enable heartbeat on a destination transport address. This value is added to the RTO of the destination transport address. This value, if present, effects all destinations.

J) Request HeartBeat

Format: REQUESTHEARTBEAT(association id, destination transport address)
-> result

Instructs the local endpoint to perform a HeartBeat on the specified destination transport address of the given association. The returned result should indicate whether the transmission of the HEARTBEAT chunk to the destination address is successful.

Mandatory attributes:

o association id - local handle to the SCTP association

o destination transport address - the transport address of the association on which a heartbeat should be issued.

K) Get SRTT Report

Format: GETSRTTREPORT(association id, destination transport address)
-> srtt result

Instructs the local SCTP to report the current SRTT measurement on the specified destination transport address of the given association. The returned result can be an integer containing the most recent SRTT in milliseconds.

Mandatory attributes:

o association id - local handle to the SCTP association

o destination transport address - the transport address of the association on which the SRTT measurement is to be reported.

L) Set Failure Threshold

Format: SETFAILURETHRESHOLD(association id, destination transport address, failure threshold)
-> result

This primitive allows the local SCTP to customize the reachability failure detection threshold 'Path.Max.Retrans' for the specified destination address.

Mandatory attributes:

o association id - local handle to the SCTP association

o destination transport address - the transport address of the association on which the failure detection threshold is to be set.

o failure threshold - the new value of 'Path.Max.Retrans' for the destination address.

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id, [,destination transport
address,] protocol parameter list)
-> result

This primitive allows the local SCTP to customize the protocol parameters.

Mandatory attributes:

o association id - local handle to the SCTP association

o protocol parameter list - The specific names and values of the protocol parameters (e.g., Association.Max.Retrans [see Section 14]) that the SCTP user wishes to customize.

Optional attributes:

o destination transport address - some of the protocol parameters may be set on a per destination transport address basis.

N) Receive unsent message

Format: RECEIVE_UNSENT(data retrieval id, buffer address, buffer size
[,stream id] [, stream sequence number] [,partial flag]
[,payload protocol-id])

o data retrieval id - The identification passed to the ULP in the failure notification.

o buffer address - the memory location indicated by the ULP to store the received message.

o buffer size - the maximum size of data to be received, in bytes.

Optional attributes:

o stream id - this is a return value that is set to indicate which stream the data was sent to.

o stream sequence number - this value is returned indicating the stream sequence number that was associated with the message.

o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number MUST

accompany this receive.  When this flag is set to 0, it
indicates
that no more deliveries will be received for this stream
sequence
number.

o  payload protocol-id - The 32 bit unsigned integer that was sent
to
be sent to the peer indicating the type of payload protocol of
the
received data.

O)  Receive unacknowledged message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer
size,
[,stream id] [, stream sequence number] [,partial flag]
[,payload protocol-id])

o  data retrieval id - The identification passed to the ULP in the
failure notification.

o  buffer address - the memory location indicated by the ULP to
store
the received message.

o  buffer size - the maximum size of data to be received, in bytes.

Optional attributes:

o  stream id - this is a return value that is set to  indicate
which
stream the data was sent to.

o  stream sequence number - this value is returned indicating the
stream sequence number that was associated with the message.

o  partial flag - if this returned flag is set to 1, then this
message is a partial delivery of the whole message.  When this
flag is set, the stream id and stream sequence number MUST
accompany this receive.  When this flag is set to 0, it
indicates
that no more deliveries will be received for this stream
sequence
number.

o  payload protocol-id - The 32 bit unsigned integer that was sent
to
be sent to the peer indicating the type of payload protocol of
the
received data.

P) Destroy SCTP instance

Format: DESTROY(local SCTP instance name)

o  local SCTP instance name - this is the value that was passed to
the application in the initialize primitive and it indicates
which
SCTP instance to be destroyed.

### 10.2 SCTP-to-ULP

It is assumed that the operating system or application environment provides a means for the SCTP to asynchronously signal the ULP process. When SCTP does signal an ULP process, certain information is passed to the ULP.

IMPLEMENTATION NOTE: In some cases this may be done through a separate socket or error channel.

A) DATA ARRIVE notification

SCTP shall invoke this notification on the ULP when a user message is successfully received and ready for retrieval.

The following may be optionally be passed with the notification:

o association id - local handle to the SCTP association

o stream id - to indicate which stream the data is received on.

B) SEND FAILURE notification

If a message can not be delivered SCTP shall invoke this notification on the ULP.

The following may be optionally be passed with the notification:

o association id - local handle to the SCTP association

o data retrieval id - an identification used to retrieve unsent and unacknowledged data.

o cause code - indicating the reason of the failure, e.g., size too large, message life-time expiration, etc.

o context - optional information associated with this message (see D in Section 10.1).

C) NETWORK STATUS CHANGE notification

When a destination transport address is marked inactive (e.g., when SCTP detects a failure), or marked active (e.g., when SCTP detects a recovery), SCTP shall invoke this notification on the ULP.

The following shall be passed with the notification:

o association id - local handle to the SCTP association

o destination transport address - This indicates the destination transport address of the peer endpoint affected by the change;

o new-status - This indicates the new status.

D) COMMUNICATION UP notification

This notification is used when SCTP becomes ready to send or receive user messages, or when a lost communication to an endpoint is restored.

IMPLEMENTATION NOTE: If ASSOCIATE primitive is implemented as a blocking function call, the association parameters are returned as a result of the ASSOCIATE primitive itself. In that case, COMMUNICATION UP notification is optional at the association initiator's side.

The following shall be passed with the notification:

- association id - local handle to the SCTP association

- status - This indicates what type of event has occurred

- destination transport address list - the complete set of transport addresses of the peer

- outbound stream count - the maximum number of streams allowed to be used in this association by the ULP

- inbound stream count - the number of streams the peer endpoint has requested with this association (this may not be the same number as 'outbound stream count').

E) COMMUNICATION LOST notification

When SCTP loses communication to an endpoint completely (e.g., via Heartbeats) or detects that the endpoint has performed an abort operation, it shall invoke this notification on the ULP.

The following shall be passed with the notification:

- association id - local handle to the SCTP association

- status - This indicates what type of event has occurred; The status may indicate a failure OR a normal termination event occurred in response to a shutdown or abort request.

The following may be passed with the notification:

- data retrieval id - an identification used to retrieve unsent and unacknowledged data.

- last-acked - the TSN last acked by that peer endpoint;

- last-sent - the TSN last sent to that peer endpoint;

F) COMMUNICATION ERROR notification

When SCTP receives an ERROR chunk from its peer and decides to notify its ULP, it can invoke this notification on the ULP.

The following can be passed with the notification:

o association id - local handle to the SCTP association

o error info - this indicates the type of error and optionally some additional information received through the ERROR chunk.

G) RESTART notification

When SCTP detects that the peer has restarted, it may send this notification to its ULP.

The following can be passed with the notification:

o association id - local handle to the SCTP association

H) SHUTDOWN COMPLETE notification

When SCTP completes the shutdown procedures (section 9.2) this notification is passed to the upper layer.

The following can be passed with the notification:

o association id - local handle to the SCTP association

11. Security Considerations

11.1 Security Objectives

As a common transport protocol designed to reliably carry time-sensitive user messages, such as billing or signaling messages for telephony services, between two networked endpoints, SCTP has the following security objectives.

- availability of reliable and timely data transport services
- integrity of the user-to-user information carried by SCTP

11.2 SCTP Responses To Potential Threats

SCTP may potentially be used in a wide variety of risk situations. It is important for operator(s) of systems running SCTP to analyze their particular situations and decide on the appropriate counter-measures.

Operators of systems running SCTP should consult [<A HREF="/rfcs/rfc2196.html">RFC2196</A>] for guidance in securing their site.

11.2.1 Countering Insider Attacks

The principles of [<A HREF="/rfcs/rfc2196.html">RFC2196</A>] should be applied to minimize the risk of theft of information or sabotage by insiders. Such procedures include publication of security policies, control of access at the physical, software, and network levels, and separation of services.

11.2.2 Protecting against Data Corruption in the Network

Where the risk of undetected errors in datagrams delivered by the lower layer transport services is considered to be too great, additional integrity protection is required. If this additional protection were provided in the application-layer, the SCTP header would remain vulnerable to deliberate integrity attacks. While the existing SCTP mechanisms for detection of packet replays are considered sufficient for normal operation, stronger protections are needed to protect SCTP when the operating environment contains significant risk of deliberate attacks from a sophisticated adversary.

In order to promote software code-reuse, to avoid re-inventing the wheel, and to avoid gratuitous complexity to SCTP, the IP Authentication Header [<A HREF="/rfcs/rfc2402.html">RFC2402</A>] SHOULD be used when the threat environment requires stronger integrity protections, but does not require confidentiality.

A widely implemented BSD Sockets API extension exists for applications to request IP security services, such as AH or ESP from an operating system kernel. Applications can use such an API to request AH whenever AH use is appropriate.

11.2.3 Protecting Confidentiality

In most cases, the risk of breach of confidentiality applies to the signaling data payload, not to the SCTP or lower-layer protocol overheads. If that is true, encryption of the SCTP user data only might be considered. As with the supplementary checksum service, user data encryption MAY be performed by the SCTP user application.

Alternately, the user application MAY use an implementation-specific API to request that the IP Encapsulating Security Payload (ESP) [<A HREF="/rfcs/rfc2406.html">RFC2406</A>] be used to provide confidentiality and integrity.

Particularly for mobile users, the requirement for confidentiality might include the masking of IP addresses and ports. In this case ESP SHOULD be used instead of application-level confidentiality. If ESP is used to protect confidentiality of SCTP traffic, an ESP cryptographic transform that includes cryptographic integrity protection MUST be used, because if there is a confidentiality threat there will also be a strong integrity threat.

Whenever ESP is in use, application-level encryption is not generally required.

Regardless of where confidentiality is provided, the ISAKMP [<A HREF="/rfcs/rfc2408.html">RFC2408</A>] and the Internet Key Exchange (IKE) [<A HREF="/rfcs/rfc2409.html">RFC2409</A>] SHOULD be used for key management.

Operators should consult [<A HREF="/rfcs/rfc2401.html">RFC2401</A>] for more information on the security services available at and immediately above the Internet

Protocol layer.

11.2.4 Protecting against Blind Denial of Service Attacks

A blind attack is one where the attacker is unable to intercept or otherwise see the content of data flows passing to and from the target SCTP node. Blind denial of service attacks may take the form of flooding, masquerade, or improper monopolization of services.

11.2.4.1 Flooding

The objective of flooding is to cause loss of service and incorrect behavior at target systems through resource exhaustion, interference with legitimate transactions, and exploitation of buffer-related software bugs. Flooding may be directed either at the SCTP node or at resources in the intervening IP Access Links or the Internet. Where the latter entities are the target, flooding will manifest itself as loss of network services, including potentially the breach of any firewalls in place.

In general, protection against flooding begins at the equipment design level, where it includes measures such as:

- avoiding commitment of limited resources before determining that the request for service is legitimate

- giving priority to completion of processing in progress over the acceptance of new work

- identification and removal of duplicate or stale queued requests for service.

- not responding to unexpected packets sent to non-unicast addresses.

Network equipment should be capable of generating an alarm and log if a suspicious increase in traffic occurs. The log should provide information such as the identity of the incoming link and source address(es) used which will help the network or SCTP system operator to take protective measures. Procedures should be in place for the operator to act on such alarms if a clear pattern of abuse emerges.

The design of SCTP is resistant to flooding attacks, particularly in its use of a four-way start-up handshake, its use of a cookie to defer commitment of resources at the responding SCTP node until the handshake is completed, and its use of a Verification Tag to prevent insertion of extraneous packets into the flow of an established association.

The IP Authentication Header and Encapsulating Security Payload might be useful in reducing the risk of certain kinds of denial of service attacks."

The use of the Host Name feature in the INIT chunk could be used to flood a target DNS server. A large backlog of DNS queries, resolving the Host Name received in the INIT chunk to IP addresses, could be accomplished by sending INIT's to multiple hosts in a given domain. In addition, an attacker could use the Host Name feature in an indirect attack on a third party by sending large numbers of INITs to random hosts containing the host name of the target. In addition to the strain on DNS resources, this could also result in large numbers of INIT ACKs being sent to the target. One method to protect against this type of attack is to verify that the IP addresses received from DNS include the source IP address of the original INIT. If the list of IP addresses received from DNS does not include the source IP address of the INIT, the endpoint MAY silently discard the INIT. This last option will not protect against the attack against the DNS.

11.2.4.2 Blind Masquerade

Masquerade can be used to deny service in several ways:

- by tying up resources at the target SCTP node to which the impersonated node has limited access. For example, the target node may by policy permit a maximum of one SCTP association with the impersonated SCTP node. The masquerading attacker may attempt to establish an association purporting to come from the impersonated node so that the latter cannot do so when it requires it.

- by deliberately allowing the impersonation to be detected, thereby provoking counter-measures which cause the impersonated node to be locked out of the target SCTP node.

- by interfering with an established association by inserting extraneous content such as a SHUTDOWN request.

SCTP reduces the risk of blind masquerade attacks through IP spoofing by use of the four-way startup handshake. Man-in-the-middle masquerade attacks are discussed in Section 11.3 below. Because the initial exchange is memoryless, no lockout mechanism is triggered by blind masquerade attacks. In addition, the INIT ACK containing the State Cookie is transmitted back to the IP address from which it received the INIT. Thus the attacker would not receive the INIT ACK containing the State Cookie. SCTP protects against insertion of extraneous packets into the flow of an established association by use of the Verification Tag.

Logging of received INIT requests and abnormalities such as unexpected INIT ACKs might be considered as a way to detect patterns of hostile activity. However, the potential usefulness of such logging must be weighed against the increased SCTP startup processing it implies, rendering the SCTP node more vulnerable to flooding attacks. Logging is pointless without the establishment of operating procedures to review and analyze the logs on a routine basis.

#### 11.2.4.3 Improper Monopolization of Services

Attacks under this heading are performed openly and legitimately by the attacker. They are directed against fellow users of the target SCTP node or of the shared resources between the attacker and the target node. Possible attacks include the opening of a large number of associations between the attacker's node and the target, or transfer of large volumes of information within a legitimately-established association.

Policy limits should be placed on the number of associations per adjoining SCTP node. SCTP user applications should be capable of detecting large volumes of illegitimate or "no-op" messages within a given association and either logging or terminating the association as a result, based on local policy.

### 11.3 Protection against Fraud and Repudiation

The objective of fraud is to obtain services without authorization and specifically without paying for them. In order to achieve this objective, the attacker must induce the SCTP user application at the target SCTP node to provide the desired service while accepting invalid billing data or failing to collect it. Repudiation is a related problem, since it may occur as a deliberate act of fraud or simply because the repudiating party kept inadequate records of service received.

Potential fraudulent attacks include interception and misuse of authorizing information such as credit card numbers, blind masquerade and replay, and man-in-the middle attacks which modify the packets passing through a target SCTP association in real time.

The interception attack is countered by the confidentiality measures discussed in Section 11.2.3 above.

Section 11.2.4.2 describes how SCTP is resistant to blind masquerade attacks, as a result of the four-way startup handshake and the Verification Tag. The Verification Tag and TSN together are protections against blind replay attacks, where the replay is into an existing association.

However, SCTP does not protect against man-in-the-middle attacks
where the attacker is able to intercept and alter the packets sent
and received in an association.  For example, the INIT ACK will
have
sufficient information sent on the wire for an adversary in the
middle to hijack an existing SCTP association.  Where a significant
possibility of such attacks is seen to exist, or where possible
repudiation is an issue, the use of the IPSEC AH service is
recommended to ensure both the integrity and the authenticity of
the
SCTP packets passed.

SCTP also provides no protection against attacks originating at or
beyond the SCTP node and taking place within the context of an
existing association.  Prevention of such attacks should be covered
by appropriate security policies at the host site, as discussed in
Section 11.2.1.

12. Recommended Transmission Control Block (TCB) Parameters

This section details a recommended set of parameters that should be
contained within the TCB for an implementation.  This section is
for
illustrative purposes and should not be deemed as requirements on
an
implementation or as an exhaustive list of all parameters inside an
SCTP TCB.  Each implementation may need its own additional
parameters
for optimization.

12.1 Parameters necessary for the SCTP instance

Associations: A list of current associations and mappings to the
data
consumers for each association.  This may be in the
form of a hash table or other implementation
dependent
structure.  The data consumers may be process
identification information such as file descriptors,
named pipe pointer, or table pointers dependent on
how
SCTP is implemented.

Secret Key:   A secret key used by this endpoint to compute the
MAC.
This SHOULD be a cryptographic quality random number
with a sufficient length.  Discussion in [<A
HREF="/rfcs/rfc1750.html">RFC1750</A>] can
be helpful in selection of the key.

Address List: The list of IP addresses that this instance has
bound.
This information is passed to one's peer(s) in INIT
and
INIT ACK chunks.

SCTP Port:    The local SCTP port number the endpoint is bound to.

12.2 Parameters necessary per association (i.e. the TCB)

Peer        : Tag value to be sent in every packet and is received

```
   Verification: in the INIT or INIT ACK chunk.
   Tag         :

   My          : Tag expected in every inbound packet and sent in the
   Verification: INIT or INIT ACK chunk.
   Tag         :

   State       : A state variable indicating what state the
association
               : is in, i.e. COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED,
               : SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED,
               : SHUTDOWN-ACK-SENT.

                 Note: No "CLOSED" state is illustrated since if a
                 association is "CLOSED" its TCB SHOULD be removed.

   Peer        : A list of SCTP transport addresses that the peer is
   Transport   : bound to.  This information is derived from the INIT
or
   Address     : INIT ACK and is used to associate an inbound packet
   List        : with a given association.  Normally this information
is
               : hashed or keyed for quick lookup and access of the
TCB.

   Primary     : This is the current primary destination transport
   Path        : address of the peer endpoint.  It may also specify a
               : source transport address on this endpoint.

   Overall     : The overall association error count.
   Error Count :

   Overall     : The threshold for this association that if the
Overall
   Error       : Error Count reaches will cause this association to be
   Threshold   : torn down.

   Peer Rwnd   : Current calculated value of the peer's rwnd.

   Next TSN    : The next TSN number to be assigned to a new DATA
chunk.
               : This is sent in the INIT or INIT ACK chunk to the
peer
               : and incremented each time a DATA chunk is assigned a
               : TSN (normally just prior to transmit or during
               : fragmentation).

   Last Rcvd   : This is the last TSN received in sequence.  This
value
   TSN         : is set initially by taking the peer's Initial TSN,
               : received in the INIT or INIT ACK chunk, and
               : subtracting one from it.

   Mapping     : An array of bits or bytes indicating which out of
   Array       : order TSN's have been received (relative to the
               : Last Rcvd TSN).  If no gaps exist, i.e. no out of
order
               : packets have been received, this array will be set to
               : all zero.  This structure may be in the form of a
               : circular buffer or bit array.
```

```
   Ack State   : This flag indicates if the next received packet
               : is to be responded to with a SACK.  This is
initialized
               : to 0.  When a packet is received it is incremented.
               : If this value reaches 2 or more, a SACK is sent and
the
               : value is reset to 0.  Note: This is used only when no
               : DATA chunks are received out of order.  When DATA
chunks
               : are out of order, SACK's are not delayed (see Section
               : 6).

   Inbound     : An array of structures to track the inbound streams.
   Streams     : Normally including the next sequence number expected
               : and possibly the stream number.

   Outbound    : An array of structures to track the outbound streams.
   Streams     : Normally including the next sequence number to
               : be sent on the stream.

   Reasm Queue : A re-assembly queue.

   Local       : The list of local IP addresses bound in to this
   Transport   : association.
   Address     :
   List        :

   Association : The smallest PMTU discovered for all of the
   PMTU        : peer's transport addresses.
```

12.3 Per Transport Address Data

For each destination transport address in the peer's address list derived from the INIT or INIT ACK chunk, a number of data elements needs to be maintained including:

```
   Error count : The current error count for this destination.

   Error       : Current error threshold for this destination i.e.
   Threshold   : what value marks the destination down if Error count
               : reaches this value.

   cwnd        : The current congestion window.

   ssthresh    : The current ssthresh value.

   RTO         : The current retransmission timeout value.

   SRTT        : The current smoothed round trip time.

   RTTVAR      : The current RTT variation.

   partial     : The tracking method for increase of cwnd when in
   bytes acked : congestion avoidance mode (see Section 6.2.2)

   state       : The current state of this destination, i.e. DOWN, UP,
               : ALLOW-HB, NO-HEARTBEAT, etc.

   PMTU        : The current known path MTU.

   Per         : A timer used by each destination.
```

Destination :
Timer       :

RTO-Pending : A flag used to track if one of the DATA chunks sent to
this address is currently being used to compute a RTT. If this flag is 0, the next DATA chunk sent to this
destination should be used to compute a RTT and this flag should be set. Every time the RTT calculation completes (i.e. the DATA chunk is SACK'd) clear this flag.

last-time : The time this destination was last sent to. This can be
used : used to determine if a HEARTBEAT is needed.

### 12.4 General Parameters Needed

Out Queue : A queue of outbound DATA chunks.

In Queue : A queue of inbound DATA chunks.

## 13. IANA Considerations

This protocol will require port reservation like TCP for the use of "well known" servers within the Internet. All current TCP ports shall be automatically reserved in the SCTP port address space. New
requests should follow IANA's current mechanisms for TCP.

This protocol may also be extended through IANA in three ways:

-- through definition of additional chunk types,
-- through definition of additional parameter types, or
-- through definition of additional cause codes within ERROR chunks

In the case where a particular ULP using SCTP desires to have its own
ports, the ULP should be responsible for registering with IANA for getting its ports assigned.

### 13.1 IETF-defined Chunk Extension

The definition and use of new chunk types is an integral part of SCTP. Thus, new chunk types are assigned by IANA through an IETF Consensus action as defined in [<A
HREF="/rfcs/rfc2434.html">RFC2434</A>].

The documentation for a new chunk code type must include the following information:

a) A long and short name for the new chunk type;

b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2;

c) A detailed definition and description of intended use of each field within the chunk, including the chunk flags if any;

d) A detailed procedural description of the use of the new chunk type
within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

## 13.2 IETF-defined Chunk Parameter Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action as defined in [<A HREF="/rfcs/rfc2434.html">RFC2434</A>]. Documentation of the chunk parameter MUST contain the following information:

a) Name of the parameter type.

b) Detailed description of the structure of the parameter field. This structure MUST conform to the general type-length-value format described in Section 3.2.1.

c) Detailed definition of each component of the parameter value.

d) Detailed description of the intended use of this parameter type, and an indication of whether and under what circumstances multiple instances of this parameter type may be found within the same chunk.

## 13.3 IETF-defined Additional Error Causes

Additional cause codes may be allocated in the range 11 to 65535 through a Specification Required action as defined in [<A HREF="/rfcs/rfc2434.html">RFC2434</A>]. Provided documentation must include the following information:

a) Name of the error condition.

b) Detailed description of the conditions under which an SCTP endpoint should issue an ERROR (or ABORT) with this cause code.

c) Expected action by the SCTP endpoint which receives an ERROR (or ABORT) chunk containing this cause code.

d) Detailed description of the structure and content of data fields which accompany this cause code.

The initial word (32 bits) of a cause code parameter MUST conform to the format shown in Section 3.3.10, i.e.:

-- first two bytes contain the cause code value
-- last two bytes contain length of the Cause Parameter.

## 13.4 Payload Protocol Identifiers

Except for value 0 which is reserved by SCTP to indicate an unspecified payload protocol identifier in a DATA chunk, SCTP will not be responsible for standardizing or verifying any payload protocol identifiers; SCTP simply receives the identifier from the upper layer and carries it with the corresponding payload data.

The upper layer, i.e., the SCTP user, SHOULD standardize any specific protocol identifier with IANA if it is so desired. The use of any specific payload protocol identifier is out of the scope of SCTP.

## 14. Suggested SCTP Protocol Parameter Values

The following protocol parameters are RECOMMENDED:

| | |
|---|---|
| RTO.Initial | - 3 seconds |
| RTO.Min | - 1 second |
| RTO.Max | - 60 seconds |
| RTO.Alpha | - 1/8 |
| RTO.Beta | - 1/4 |
| Valid.Cookie.Life | - 60 seconds |
| Association.Max.Retrans | - 10 attempts |
| Path.Max.Retrans | - 5 attempts (per destination address) |
| Max.Init.Retransmits | - 8 attempts |
| HB.interval | - 30 seconds |

IMPLEMENTATION NOTE: The SCTP implementation may allow ULP to customize some of these protocol parameters (see Section 10).

Note: RTO.Min SHOULD be set as recommended above.

## 15. Acknowledgements

The authors wish to thank Mark Allman, R.J. Atkinson, Richard Band, Scott Bradner, Steve Bellovin, Peter Butler, Ram Dantu, R. Ezhirpavai, Mike Fisk, Sally Floyd, Atsushi Fukumoto, Matt Holdrege, Henry Houh, Christian Huitema, Gary Lehecka, Jonathan Lee, David Lehmann, John Loughney, Daniel Luan, Barry Nagelberg, Thomas Narten, Erik Nordmark, Lyndon Ong, Shyamal Prasad, Kelvin Porter, Heinz Prantner, Jarno Rajahalme, Raymond E. Reeves, Renee Revis, Ivan Arias Rodriguez, A. Sankar, Greg Sidebottom, Brian Wyld, La Monte Yarroll, and many others for their invaluable comments.

## 16. Authors' Addresses

Randall R. Stewart
24 Burning Bush Trail.
Crystal Lake, IL 60012
USA

Phone: +1-815-477-2127
EMail: <A HREF="mailto:rrs@cisco.com">rrs@cisco.com</A>

Qiaobing Xie
Motorola, Inc.
1501 W. Shure Drive, #2309
Arlington Heights, IL 60004
USA

Phone: +1-847-632-3028
EMail: <A HREF="mailto:qxie1@email.mot.com">qxie1@email.mot.com</A>

Ken Morneault

Cisco Systems Inc.
13615 Dulles Technology Drive
Herndon, VA. 20171
USA

Phone: +1-703-484-3323
EMail: <A HREF="mailto:kmorneau@cisco.com">kmorneau@cisco.com</A>

Chip Sharp
Cisco Systems Inc.
7025 Kit Creek Road
Research Triangle Park, NC  27709
USA

Phone: +1-919-392-3121
EMail: <A HREF="mailto:chsharp@cisco.com">chsharp@cisco.com</A>

Hanns Juergen Schwarzbauer
SIEMENS AG
Hofmannstr. 51
81359 Munich
Germany

Phone: +49-89-722-24236
EMail: <A
HREF="mailto:HannsJuergen.Schwarzbauer@icn.siemens.de">HannsJuergen.Sc
hwarzbauer@icn.siemens.de</A>

Tom Taylor
Nortel Networks
1852 Lorraine Ave.
Ottawa, Ontario
Canada K1H 6Z8

Phone: +1-613-736-0961
EMail: <A
HREF="mailto:taylor@nortelnetworks.com">taylor@nortelnetworks.com</A>

Ian Rytina
Ericsson Australia
37/360 Elizabeth Street
Melbourne, Victoria 3000
Australia

Phone: +61-3-9301-6164
EMail: <A
HREF="mailto:ian.rytina@ericsson.com">ian.rytina@ericsson.com</A>

Malleswar Kalla
Telcordia Technologies
3 Corporate Place
PYA-2J-341
Piscataway, NJ  08854
USA

Phone: +1-732-699-3728
EMail: <A
HREF="mailto:mkalla@telcordia.com">mkalla@telcordia.com</A>

Lixia Zhang
UCLA Computer Science Department

4531G Boelter Hall
Los Angeles, CA 90095-1596
USA

Phone: +1-310-825-2695
EMail: <A HREF="mailto:lixia@cs.ucla.edu">lixia@cs.ucla.edu</A>

Vern Paxson
ACIRI
1947 Center St., Suite 600,
Berkeley, CA 94704-1198
USA

Phone: +1-510-666-2882
EMail: <A HREF="mailto:vern@aciri.org">vern@aciri.org</A>

## 17. References

[<A HREF="/rfcs/rfc768.html">RFC768</A>] Postel, J. (ed.), "User Datagram Protocol", STD 6, RFC 768, August 1980.

[<A HREF="/rfcs/rfc793.html">RFC793</A>] Postel, J. (ed.), "Transmission Control Protocol", STD 7, <A HREF="/rfcs/rfc793.html">RFC 793</A>, September 1981.

[<A HREF="/rfcs/rfc1123.html">RFC1123</A>] Braden, R., "Requirements for Internet hosts - application and support", STD 3, <A HREF="/rfcs/rfc1123.html">RFC 1123</A>, October 1989.

[<A HREF="/rfcs/rfc1191.html">RFC1191</A>] Mogul, J. and S. Deering, "Path MTU Discovery", <A HREF="/rfcs/rfc1191.html">RFC 1191</A>, November 1990.

[<A HREF="/rfcs/rfc1700.html">RFC1700</A>] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994.

[<A HREF="/rfcs/rfc1981.html">RFC1981</A>] McCann, J., Deering, S. and J. Mogul, "Path MTU Discovery for IP version 6", <A HREF="/rfcs/rfc1981.html">RFC 1981</A>, August 1996.

[<A HREF="/rfcs/rfc1982.html">RFC1982</A>] Elz, R. and R. Bush, "Serial Number Arithmetic", <A HREF="/rfcs/rfc1982.html">RFC 1982</A>, August 1996.

[<A HREF="/rfcs/rfc2026.html">RFC2026</A>] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, <A HREF="/rfcs/rfc2026.html">RFC 2026</A>, October 1996.

[<A HREF="/rfcs/rfc2119.html">RFC2119</A>] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, <A HREF="/rfcs/rfc2119.html">RFC 2119</A>, March 1997.

[<A HREF="/rfcs/rfc2401.html">RFC2401</A>] Kent, S. and R. Atkinson, "Security Architecture for the

Internet Protocol", <A HREF="/rfcs/rfc2401.html">RFC 2401</A>, November 1998.

[<A HREF="/rfcs/rfc2402.html">RFC2402</A>] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[<A HREF="/rfcs/rfc2406.html">RFC2406</A>] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", <A HREF="/rfcs/rfc2406.html">RFC 2406</A>, November 1998.

[<A HREF="/rfcs/rfc2408.html">RFC2408</A>] Maughan, D., Schertler, M., Schneider, M. and J. Turner, "Internet Security Association and Key Management Protocol", <A HREF="/rfcs/rfc2408.html">RFC 2408</A>, November 1998.

[<A HREF="/rfcs/rfc2409.html">RFC2409</A>] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", <A HREF="/rfcs/rfc2409.html">RFC 2409</A>, November 1998.

[<A HREF="/rfcs/rfc2434.html">RFC2434</A>] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, <A HREF="/rfcs/rfc2434.html">RFC 2434</A>, October 1998.

[<A HREF="/rfcs/rfc2460.html">RFC2460</A>] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", <A HREF="/rfcs/rfc2460.html">RFC 2460</A>, December 1998.

[<A HREF="/rfcs/rfc2581.html">RFC2581</A>] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", <A HREF="/rfcs/rfc2581.html">RFC 2581</A>, April 1999.

18. Bibliography

[ALLMAN99] Allman, M. and Paxson, V., "On Estimating End-to-End Network Path Properties", Proc. SIGCOMM'99, 1999.

[FALL96] Fall, K. and Floyd, S., Simulation-based Comparisons of Tahoe, Reno, and SACK TCP, Computer Communications Review, V. 26 N. 3, July 1996, pp. 5-21.

[<A HREF="/rfcs/rfc1750.html">RFC1750</A>] Eastlake, D. (ed.), "Randomness Recommendations for Security", <A HREF="/rfcs/rfc1750.html">RFC 1750</A>, December 1994.

[<A HREF="/rfcs/rfc1950.html">RFC1950</A>] Deutsch P. and J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", <A HREF="/rfcs/rfc1950.html">RFC 1950</A>, May 1996.

[<A HREF="/rfcs/rfc2104.html">RFC2104</A>] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-

Hashing for Message Authentication", <A HREF="/rfcs/rfc2104.html">RFC 2104</A>, March 1997.

[<A HREF="/rfcs/rfc2196.html">RFC2196</A>] Fraser, B., "Site Security Handbook", FYI 8, <A HREF="/rfcs/rfc2196.html">RFC 2196</A>, September 1997.

[<A HREF="/rfcs/rfc2522.html">RFC2522</A>] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", <A HREF="/rfcs/rfc2522.html">RFC 2522</A>, March 1999.

[SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and Anderson, T., "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Communication Review, 29(5), October 1999.

Appendix A: Explicit Congestion Notification

ECN (Ramakrishnan, K., Floyd, S., "Explicit Congestion Notification", <A HREF="/rfcs/rfc2481.html">RFC 2481</A>, January 1999) describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss. This is an optional feature that an implementation MAY choose to add to SCTP. This appendix details the minor differences implementers will need to be aware of if they choose to implement this feature. In general <A HREF="/rfcs/rfc2481.html">RFC 2481</A> should be followed with the following exceptions.

Negotiation:

<A HREF="/rfcs/rfc2481.html">RFC2481</A> details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection. The sender of the SYN sets two bits in the TCP flags, and the sender of the SYN-ACK sets only 1 bit. The reasoning behind this is to assure both sides are truly ECN capable. For SCTP this is not necessary. To indicate that an endpoint is ECN capable an endpoint SHOULD add to the INIT and or INIT ACK chunk the TLV reserved for ECN. This TLV contains no parameters, and thus has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Parameter Type = 32768      |     Parameter Length = 4      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ECN-Echo:

<A HREF="/rfcs/rfc2481.html">RFC 2481</A> details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network. For SCTP this same indication is made by including the ECNE chunk. This chunk contains one data element, i.e. the lowest TSN associated with the IP datagram marked with the CE bit, and looks as follows:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    | Chunk Type=12 | Flags=00000000|    Chunk Length = 8
|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    |                      Lowest TSN Number
|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Note: The ECNE is considered a Control chunk.

CWR:

<A HREF="/rfcs/rfc2481.html">RFC 2481</A> details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window. This is termed the CWR bit. For SCTP the same indication is made by including the CWR chunk. This chunk contains one data element, i.e. the TSN number that was sent in the ECNE chunk. This element represents the lowest TSN number in the datagram that was originally marked with the CE bit.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    | Chunk Type=13 | Flags=00000000|    Chunk Length = 8
|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
    |                      Lowest TSN Number
|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
+
```

Note: The CWR is considered a Control chunk.

Appendix B Alder 32 bit checksum calculation

The Adler-32 checksum calculation given in this appendix is copied from [<A HREF="/rfcs/rfc1950.html">RFC1950</A>].

Adler-32 is composed of two sums accumulated per byte: s1 is the sum of all bytes, s2 is the sum of all s1 values. Both sums are done modulo 65521. s1 is initialized to 1, s2 to zero. The Adler-32 checksum is stored as s2*65536 + s1 in network byte order.

The following C code computes the Adler-32 checksum of a data buffer.
It is written for clarity, not for speed. The sample code is in the
ANSI C programming language. Non C users may find it easier to read
with these hints:

```
   &amp;      Bitwise AND operator.
   &gt;&gt;     Bitwise right shift operator.  When applied to an
          unsigned quantity, as here, right shift inserts zero bit(s)
          at the left.
   &lt;&lt;     Bitwise left shift operator.  Left shift inserts zero
          bit(s) at the right.
   ++     "n++" increments the variable n.
   %      modulo operator: a % b is the remainder of a divided by b.
    #define BASE 65521 /* largest prime smaller than 65536 */
    /*
      Update a running Adler-32 checksum with the bytes buf[0..len-1]
      and return the updated checksum.  The Adler-32 checksum should
be
      initialized to 1.

       Usage example:

         unsigned long adler = 1L;

         while (read_buffer(buffer, length) != EOF) {
           adler = update_adler32(adler, buffer, length);
         }
         if (adler != original_adler) error();
      */
      unsigned long update_adler32(unsigned long adler,
         unsigned char *buf, int len)
      {
        unsigned long s1 = adler &amp; 0xffff;
        unsigned long s2 = (adler &gt;&gt; 16) &amp; 0xffff;
        int n;

        for (n = 0; n &lt; len; n++) {
          s1 = (s1 + buf[n]) % BASE;
          s2 = (s2 + s1)     % BASE;
        }
        return (s2 &lt;&lt; 16) + s1;
      }

      /* Return the adler32 of the bytes buf[0..len-1] */
      unsigned long adler32(unsigned char *buf, int len)
      {
        return update_adler32(1L, buf, len);
      }
```

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the
Internet Society.

# RFC2018

Network Working Group M. Mathis
Request for Comments: 2018 J. Mahdavi
Category: Standards Track PSC
S. Floyd
LBNL
A. Romanow
Sun Microsystems
October 1996

TCP Selective Acknowledgment Options

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

TCP may experience poor performance when multiple packets are lost from one window of data. With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time. An aggressive sender could choose to retransmit packets early, but such retransmitted segments may have already been successfully received.

A Selective Acknowledgment (SACK) mechanism, combined with a selective repeat retransmission policy, can help to overcome these limitations. The receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments.

This memo proposes an implementation of SACK and discusses its performance and related issues.

Acknowledgements

Much of the text in this document is taken directly from RFC1072 "TCP Extensions for Long-Delay Paths" by Bob Braden and Van Jacobson. The authors would like to thank Kevin Fall (LBNL), Christian Huitema (INRIA), Van Jacobson (LBNL), Greg Miller (MITRE), Greg Minshall (Ipsilon), Lixia Zhang (XEROX PARC and UCLA), Dave Borman (BSDI), Allison Mankin (ISI) and others for their review and constructive comments.

1. Introduction

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. TCP [Postel81] uses a cumulative

acknowledgment scheme in which received segments that are not at the
left edge of the receive window are not acknowledged. This forces the sender to either wait a roundtrip time to find out about each lost packet, or to unnecessarily retransmit segments which have been
correctly received [Fall95]. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput.

Selective Acknowledgment (SACK) is a strategy which corrects this behavior in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost.

Several transport protocols, including NETBLT [Clark87], XTP [Strayer92], RDP [Velten84], NADIR [Huitema81], and VMTP [Cheriton88]
have used selective acknowledgment. There is some empirical evidence
in favor of selective acknowledgments -- simple experiments with RDP
have shown that disabling the selective acknowledgment facility greatly increases the number of retransmitted segments over a lossy,
high-delay Internet path [Partridge87]. A recent simulation study by
Kevin Fall and Sally Floyd [Fall95], demonstrates the strength of TCP
with SACK over the non-SACK Tahoe and Reno TCP implementations.

RFC1072 [VJ88] describes one possible implementation of SACK options
for TCP. Unfortunately, it has never been deployed in the Internet,
as there was disagreement about how SACK options should be used in conjunction with the TCP window shift option (initially described RFC1072 and revised in [Jacobson92]).

We propose slight modifications to the SACK options as proposed in RFC1072. Specifically, sending a selective acknowledgment for the most recently received data reduces the need for long SACK options [Keshav94, Mathis95]. In addition, the SACK option now carries full
32 bit sequence numbers. These two modifications represent the only
changes to the proposal in RFC1072. They make SACK easier to implement and address concerns about robustness.

The selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a
SYN segment to indicate that the SACK option can be used once the connection is established. The other is the SACK option itself, which may be sent over an established connection once permission has
been given by SACK-permitted.

The SACK option is to be included in a segment sent from a TCP that

is receiving data to the TCP that is sending that data; we will
refer
to these TCP's as the data receiver and the data sender,
respectively.  We will consider a particular simplex data flow; any
data flowing in the reverse direction over the same connection can
be
treated independently.

2.  Sack-Permitted Option

This two-byte option may be sent in a SYN by a TCP that has been
extended to receive (and presumably process) the SACK option once
the
connection has opened.  It MUST NOT be sent on non-SYN segments.

```
    TCP Sack-Permitted Option:

    Kind: 4

    +---------+---------+
    | Kind=4  | Length=2|
    +---------+---------+
```

3.  Sack Option Format

The SACK option is to be used to convey extended acknowledgment
information from the receiver to the sender over an established TCP
connection.

```
    TCP SACK Option:

    Kind: 5

    Length: Variable

                      +--------+--------+
                      | Kind=5 | Length |
    +--------+--------+--------+--------+
    |      Left Edge of 1st Block       |
    +--------+--------+--------+--------+
    |      Right Edge of 1st Block      |
    +--------+--------+--------+--------+
    |                                   |
    /            . . .                  /
    |                                   |
    +--------+--------+--------+--------+
    |      Left Edge of nth Block       |
    +--------+--------+--------+--------+
    |      Right Edge of nth Block      |
    +--------+--------+--------+--------+
```

The SACK option is to be sent by a data receiver to inform the data
sender of non-contiguous blocks of data that have been received and
queued.  The data receiver awaits the receipt of data (perhaps by
means of retransmissions) to fill the gaps in sequence space
between
received blocks.  When missing segments are received, the data
receiver acknowledges the data normally by advancing the left
window
edge in the Acknowledgement Number Field of the TCP header.  The
SACK

option does not change the meaning of the Acknowledgement Number field.

This option contains a list of some of the blocks of contiguous sequence space occupied by data that has been received and queued within the window.

Each contiguous block of data queued at the data receiver is defined in the SACK option by two 32-bit unsigned integers in network byte order:

* Left Edge of Block

  This is the first sequence number of this block.

* Right Edge of Block

  This is the sequence number immediately following the last sequence number of this block.

Each block represents received bytes of data that are contiguous and isolated; that is, the bytes just below the block, (Left Edge of Block - 1), and just above the block, (Right Edge of Block), have not been received.

A SACK option that specifies n blocks will have a length of 8*n+2 bytes, so the 40 bytes available for TCP options can specify a maximum of 4 blocks. It is expected that SACK will often be used in conjunction with the Timestamp option used for RTTM [Jacobson92], which takes an additional 10 bytes (plus two bytes of padding); thus a maximum of 3 SACK blocks will be allowed in this case.

The SACK option is advisory, in that, while it notifies the data sender that the data receiver has received the indicated segments, the data receiver is permitted to later discard data which have been reported in a SACK option. A discussion appears below in Section 8 of the consequences of advisory SACK, in particular that the data receiver may renege, or drop already SACKed data.

## 4. Generating Sack Options: Data Receiver Behavior

If the data receiver has received a SACK-Permitted option on the SYN for this connection, the data receiver MAY elect to generate SACK options as described below. If the data receiver generates SACK options under any circumstance, it SHOULD generate them under all permitted circumstances. If the data receiver has not received a SACK-Permitted option for a given connection, it MUST NOT send SACK options on that connection.

If sent at all, SACK options SHOULD be included in all ACKs which do not ACK the highest sequence number in the data receiver's queue. In this situation the network has lost or mis-ordered data, such that the receiver holds non-contiguous data in its queue. RFC 1122, Section 4.2.2.21, discusses the reasons for the receiver to send ACKs in response to additional segments received in this state. The receiver SHOULD send an ACK for every valid segment that arrives containing new data, and each of these "duplicate" ACKs SHOULD bear a SACK option.

If the data receiver chooses to send a SACK option, the following rules apply:

* The first SACK block (i.e., the one immediately following the kind and length fields in the option) MUST specify the contiguous block of data containing the segment which triggered this ACK, unless that segment advanced the Acknowledgment Number field in the header. This assures that the ACK with the SACK option reflects the most recent change in the data receiver's buffer queue.

* The data receiver SHOULD include as many distinct SACK blocks as possible in the SACK option. Note that the maximum available option space may not be sufficient to report all blocks present in the receiver's queue.

* The SACK option SHOULD be filled out by repeating the most recently reported SACK blocks (based on first SACK blocks in previous SACK options) that are not subsets of a SACK block already included in the SACK option being constructed. This assures that in normal operation, any segment remaining part of a non-contiguous block of data held by the data receiver is reported in at least three successive SACK options, even for large-window TCP implementations [RFC1323]). After the first SACK block, the following SACK blocks in the SACK option may be listed in arbitrary order.

It is very important that the SACK option always reports the block containing the most recently received segment, because this provides the sender with the most up-to-date information about the state of the network and the data receiver's queue.

5. Interpreting the Sack Option and Retransmission Strategy: Data Sender Behavior

When receiving an ACK containing a SACK option, the data sender SHOULD record the selective acknowledgment for future reference. The data sender is assumed to have a retransmission queue that contains the segments that have been transmitted but not yet acknowledged, in sequence-number order. If the data sender performs re-packetization before retransmission, the block boundaries in a SACK option that it receives may not fall on boundaries of segments in the retransmission queue; however, this does not pose a serious difficulty for the sender.

One possible implementation of the sender's behavior is as follows. Let us suppose that for each segment in the retransmission queue there is a (new) flag bit "SACKed", to be used to indicate that this particular segment has been reported in a SACK option.

When an acknowledgment segment arrives containing a SACK option, the data sender will turn on the SACKed bits for segments that have been selectively acknowledged. More specifically, for each block in the SACK option, the data sender will turn on the SACKed flags for all segments in the retransmission queue that are wholly contained within that block. This requires straightforward sequence number comparisons.

After the SACKed bit is turned on (as the result of processing a received SACK option), the data sender will skip that segment during any later retransmission. Any segment that has the SACKed bit turned off and is less than the highest SACKed segment is available for retransmission.

After a retransmit timeout the data sender SHOULD turn off all of the SACKed bits, since the timeout might indicate that the data receiver has reneged. The data sender MUST retransmit the segment at the left edge of the window after a retransmit timeout, whether or not the SACKed bit is on for that segment. A segment will not be dequeued and its buffer freed until the left window edge is advanced over it.

## 5.1 Congestion Control Issues

This document does not attempt to specify in detail the congestion control algorithms for implementations of TCP with SACK. However, the congestion control algorithms present in the de facto standard TCP implementations MUST be preserved [Stevens94]. In particular, to preserve robustness in the presence of packets reordered by the network, recovery is not triggered by a single ACK reporting out-of-order packets at the receiver. Further, during recovery the data sender limits the number of segments sent in response to each ACK. Existing implementations limit the data sender to sending one segment during Reno-style fast recovery, or to two segments during slow-start [Jacobson88]. Other aspects of congestion control, such as reducing the congestion window in response to congestion, must similarly be preserved.

The use of time-outs as a fall-back mechanism for detecting dropped packets is unchanged by the SACK option. Because the data receiver is allowed to discard SACKed data, when a retransmit timeout occurs the data sender MUST ignore prior SACK information in determining which data to retransmit.

Future research into congestion control algorithms may take advantage of the additional information provided by SACK. One such area for future research concerns modifications to TCP for a wireless or satellite environment where packet loss is not necessarily an indication of congestion.

## 6. Efficiency and Worst Case Behavior

If the return path carrying ACKs and SACK options were lossless, one block per SACK option packet would always be sufficient. Every segment arriving while the data receiver holds discontinuous data would cause the data receiver to send an ACK with a SACK option containing the one altered block in the receiver's queue. The data sender is thus able to construct a precise replica of the receiver's queue by taking the union of all the first SACK blocks.

Since the return path is not lossless, the SACK option is defined to include more than one SACK block in a single packet. The redundant blocks in the SACK option packet increase the robustness of SACK delivery in the presence of lost ACKs. For a receiver that is also using the time stamp option [Jacobson92], the SACK option has room to include three SACK blocks. Thus each SACK block will generally be repeated at least three times, if necessary, once in each of three successive ACK packets. However, if all of the ACK packets reporting a particular SACK block are dropped, then the sender might assume that the data in that SACK block has not been received, and unnecessarily retransmit those segments.

The deployment of other TCP options may reduce the number of available SACK blocks to 2 or even to 1. This will reduce the redundancy of SACK delivery in the presence of lost ACKs. Even so, the exposure of TCP SACK in regard to the unnecessary retransmission of packets is strictly less than the exposure of current implementations of TCP. The worst-case conditions necessary for the sender to needlessly retransmit data is discussed in more detail in a separate document [Floyd96].

Older TCP implementations which do not have the SACK option will not be unfairly disadvantaged when competing against SACK-capable TCPs. This issue is discussed in more detail in [Floyd96].

## 7. Sack Option Examples

The following examples attempt to demonstrate the proper behavior
of
SACK generation by the data receiver.

Assume the left window edge is 5000 and that the data transmitter sends a burst of 8 segments, each containing 500 data bytes.

Case 1: The first 4 segments are received but the last 4 are dropped.

The data receiver will return a normal TCP ACK segment acknowledging sequence number 7000, with no SACK option.

Case 2: The first segment is dropped but the remaining 7 are received.

Upon receiving each of the last seven packets, the data receiver will return a TCP ACK segment that acknowledges sequence number 5000 and contains a SACK option specifying one block of queued data:

| Triggering Segment | ACK | Left Edge | Right Edge |
|---|---|---|---|
| 5000 | (lost) | | |
| 5500 | 5000 | 5500 | 6000 |
| 6000 | 5000 | 5500 | 6500 |
| 6500 | 5000 | 5500 | 7000 |
| 7000 | 5000 | 5500 | 7500 |
| 7500 | 5000 | 5500 | 8000 |
| 8000 | 5000 | 5500 | 8500 |
| 8500 | 5000 | 5500 | 9000 |

Case 3: The 2nd, 4th, 6th, and 8th (last) segments are dropped.

The data receiver ACKs the first packet normally. The third, fifth, and seventh packets trigger SACK options as follows:

| Triggering Segment | ACK | First Block Left Edge | First Block Right Edge | 2nd Block Left Edge | 2nd Block Right Edge | 3rd Block Left Edge | 3rd Block Right Edge |
|---|---|---|---|---|---|---|---|
| 5000 | 5500 | | | | | | |
| 5500 | (lost) | | | | | | |
| 6000 | 5500 | 6000 | 6500 | | | | |
| 6500 | (lost) | | | | | | |
| 7000 | 5500 | 7000 | 7500 | 6000 | 6500 | | |
| 7500 | (lost) | | | | | | |
| 8000 | 5500 | 8000 | 8500 | 7000 | 7500 | 6000 | 6500 |
| 8500 | (lost) | | | | | | |

Suppose at this point, the 4th packet is received out of order. (This could either be because the data was badly misordered in
the
network, or because the 2nd packet was retransmitted and lost,
and
then the 4th packet was retransmitted). At this point the data receiver has only two SACK blocks to report. The data receiver replies with the following Selective Acknowledgment:

| Triggering Segment | ACK | First Block Left Edge | First Block Right Edge | 2nd Block Left Edge | 2nd Block Right Edge | 3rd Block Left Edge | 3rd Block Right Edge |
|---|---|---|---|---|---|---|---|
| 6500 | 5500 | 6000 | 7500 | 8000 | 8500 | | |

Suppose at this point, the 2nd segment is received. The data receiver then replies with the following Selective Acknowledgment:

| Triggering Segment | ACK | First Block Left Edge | First Block Right Edge | 2nd Block Left Edge | 2nd Block Right Edge | 3rd Block Left Edge | 3rd Block Right Edge |
|---|---|---|---|---|---|---|---|
| 5500 | 7500 | 8000 | 8500 | | | | |

## 8. Data Receiver Reneging

Note that the data receiver is permitted to discard data in its queue that has not been acknowledged to the data sender, even if the data has already been reported in a SACK option. Such discarding of SACKed packets is discouraged, but may be used if the receiver runs out of buffer space.

The data receiver MAY elect not to keep data which it has reported in a SACK option. In this case, the receiver SACK generation is additionally qualified:

* The first SACK block MUST reflect the newest segment. Even if the newest segment is going to be discarded and the receiver has already discarded adjacent segments, the first SACK block MUST report, at a minimum, the left and right edges of the newest segment.

* Except for the newest segment, all SACK blocks MUST NOT report any old data which is no longer actually held by the receiver.

Since the data receiver may later discard data reported in a SACK option, the sender MUST NOT discard data before it is acknowledged by the Acknowledgment Number field in the TCP header.

## 9. Security Considerations

This document neither strengthens nor weakens TCP's current security properties.

## 10. References

[Cheriton88] Cheriton, D., "VMTP: Versatile Message Transaction Protocol", RFC 1045, Stanford University, February 1988.

[Clark87] Clark, D., Lambert, M., and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", RFC 998, MIT, March 1987.

[Fall95] Fall, K. and Floyd, S., "Comparisons of Tahoe, Reno, and

Sack TCP", ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z, December 1995.

[Floyd96] Floyd, S., "Issues of TCP with SACK", ftp://ftp.ee.lbl.gov/papers/issues_sa.ps.Z, January 1996.

[Huitema81] Huitema, C., and Valet, I., An Experiment on High Speed File Transfer using Satellite Links, 7th Data Communication Symposium, Mexico, October 1981.

[Jacobson88] Jacobson, V., "Congestion Avoidance and Control", Proceedings of SIGCOMM '88, Stanford, CA., August 1988.

[Jacobson88}, Jacobson, V. and R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, October 1988.

[Jacobson92] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[Keshav94] Keshav, presentation to the Internet End-to-End Research Group, November 1994.

[Mathis95] Mathis, M., and Mahdavi, J., TCP Forward Acknowledgment Option, presentation to the Internet End-to-End Research Group, June 1995.

[Partridge87] Partridge, C., "Private Communication", February 1987.

[Postel81] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[Stevens94] Stevens, W., TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

[Strayer92] Strayer, T., Dempsey, B., and Weaver, A., XTP -- the xpress transfer protocol. Addison-Wesley Publishing Company, 1992.

[Velten84] Velten, D., Hinden, R., and J. Sax, "Reliable Data Protocol", RFC 908, BBN, July 1984.

11. Authors' Addresses

Matt Mathis and Jamshid Mahdavi
Pittsburgh Supercomputing Center
4400 Fifth Ave
Pittsburgh, PA 15213
mathis@psc.edu
mahdavi@psc.edu

Sally Floyd
Lawrence Berkeley National Laboratory
One Cyclotron Road
Berkeley, CA 94720
floyd@ee.lbl.gov

Allyn Romanow

Sun Microsystems, Inc.
2550 Garcia Ave., MPK17-202
Mountain View, CA 94043
allyn@eng.sun.com

# RFC2581

Network Working Group
Request for Comments: 2581
Obsoletes: 2001
Category: Standards Track

M. Allman
NASA Glenn/Sterling Software
V. Paxson
ACIRI / ICSI
W. Stevens
Consultant

April 1999

TCP Congestion Control

Status of this Memo



Copyright Notice



Abstract

This document defines TCP's four intertwined congestion control algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery. In addition, the document specifies how TCP should begin transmission after a relatively long idle period, as well as discussing various acknowledgment generation methods.

1. Introduction

This document specifies four TCP [Pos81] congestion control algorithms: slow start, congestion avoidance, fast retransmit and fast recovery. These algorithms were devised in [Jac88] and [Jac90]. Their use with TCP is standardized in [Bra89].

This document is an update of [Ste97]. In addition to specifying the congestion control algorithms, this document specifies what TCP connections should do after a relatively long idle period, as well as specifying and clarifying some of the issues pertaining to TCP ACK generation.

Note that [Ste94] provides examples of these algorithms in action and
[WS95] provides an explanation of the source code for the BSD implementation of these algorithms.

This document is organized as follows. Section 2 provides various definitions which will be used throughout the document. Section 3 provides a specification of the congestion control algorithms. Section 4 outlines concerns related to the congestion control algorithms and finally, section 5 outlines security considerations.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [Bra97].

## 2. Definitions

This section provides the definition of several terms that will be used throughout the remainder of this document.

SEGMENT:
A segment is ANY TCP/IP data or acknowledgment packet (or both).

SENDER MAXIMUM SEGMENT SIZE (SMSS): The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [MD90] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

RECEIVER MAXIMUM SEGMENT SIZE (RMSS): The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, 536 bytes
[Bra89]. The size does not include the TCP/IP headers and options.

FULL-SIZED SEGMENT: A segment that contains the maximum number of data bytes permitted (i.e., a segment containing SMSS bytes of data).

RECEIVER WINDOW (rwnd) The most recently advertised receiver window.

CONGESTION WINDOW (cwnd): A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP MUST NOT
send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

INITIAL WINDOW (IW): The initial window is the size of the sender's
congestion window after the three-way handshake is completed.

LOSS WINDOW (LW): The loss window is the size of the congestion window after a TCP sender detects loss using its retransmission timer.

RESTART WINDOW (RW): The restart window is the size of the congestion window after a TCP restarts transmission after an idle period (if the slow start algorithm is used; see section 4.1 for more discussion).

FLIGHT SIZE: The amount of data that has been sent but not yet acknowledged.

## 3. Congestion Control Algorithms

This section defines the four congestion control algorithms: slow start, congestion avoidance, fast retransmit and fast recovery, developed in [Jac88] and [Jac90]. In some situations it may be beneficial for a TCP sender to be more conservative than the algorithms allow, however a TCP MUST NOT be more aggressive than the following algorithms allow (that is, MUST NOT send data when the value of cwnd computed by the following algorithms would not allow the data to be sent).

### 3.1 Slow Start and Congestion Avoidance

The slow start and congestion avoidance algorithms MUST be used by a TCP sender to control the amount of outstanding data being injected into the network. To implement these algorithms, two variables are added to the TCP per-connection state. The congestion window (cwnd) is a sender-side limit on the amount of data the sender can transmit into the network before receiving an acknowledgment (ACK), while the receiver's advertised window (rwnd) is a receiver-side limit on the amount of outstanding data. The minimum of cwnd and rwnd governs data transmission.

Another state variable, the slow start threshold (ssthresh), is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission, as discussed below.

Beginning transmission into a network with unknown conditions requires TCP to slowly probe the network to determine the available capacity, in order to avoid congesting the network with an inappropriately large burst of data. The slow start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

IW, the initial value of cwnd, MUST be less than or equal to 2*SMSS bytes and MUST NOT be more than 2 segments.

We note that a non-standard, experimental TCP extension allows that a TCP MAY use a larger initial window (IW), as defined in equation 1 [AFP98]:

$$IW = \min (4*SMSS, \max (2*SMSS, 4380 \text{ bytes})) \qquad (1)$$

With this extension, a TCP sender MAY use a 3 or 4 segment initial window, provided the combined size of the segments does not exceed 4380 bytes. We do NOT allow this change as part of the standard defined by this document. However, we include discussion of (1) in the remainder of this document as a guideline for those experimenting with the change, rather than conforming to the present standards for TCP congestion control.

The initial value of ssthresh MAY be arbitrarily high (for example, some implementations use the size of the advertised window), but it may be reduced in response to congestion. The slow start algorithm is used when cwnd < ssthresh, while the congestion avoidance algorithm is used when cwnd > ssthresh. When cwnd and ssthresh are equal the sender may use either slow start or congestion avoidance.

During slow start, a TCP increments cwnd by at most SMSS bytes for each ACK received that acknowledges new data. Slow start ends when cwnd exceeds ssthresh (or, optionally, when it reaches it, as noted above) or when congestion is observed.

During congestion avoidance, cwnd is incremented by 1 full-sized segment per round-trip time (RTT). Congestion avoidance continues until congestion is detected. One formula commonly used to update cwnd during congestion avoidance is given in equation 2:

cwnd += SMSS*SMSS/cwnd                     (2)

This adjustment is executed on every incoming non-duplicate ACK. Equation (2) provides an acceptable approximation to the underlying principle of increasing cwnd by 1 full-sized segment per RTT. (Note that for a connection in which the receiver acknowledges every data segment, (2) proves slightly more aggressive than 1 segment per RTT, and for a receiver acknowledging every-other packet, (2) is less aggressive.)

Implementation Note: Since integer arithmetic is usually used in TCP implementations, the formula given in equation 2 can fail to increase cwnd when the congestion window is very large (larger than SMSS*SMSS). If the above formula yields 0, the result SHOULD be rounded up to 1 byte.

Implementation Note: older implementations have an additional additive constant on the right-hand side of equation (2). This is incorrect and can actually lead to diminished performance [PAD+98].

Another acceptable way to increase cwnd during congestion avoidance is to count the number of bytes that have been acknowledged by ACKs for new data. (A drawback of this implementation is that it requires maintaining an additional state variable.) When the number of bytes acknowledged reaches cwnd, then cwnd can be incremented by up to SMSS bytes. Note that during congestion avoidance, cwnd MUST NOT be

increased by more than the larger of either 1 full-sized segment per
RTT, or the value computed using equation 2.

Implementation Note: some implementations maintain cwnd in units of bytes, while others in units of full-sized segments. The latter will find equation (2) difficult to use, and may prefer to use the counting approach discussed in the previous paragraph.

When a TCP sender detects segment loss using the retransmission timer, the value of ssthresh MUST be set to no more than the value given in equation 3:

$$ssthresh = max (FlightSize / 2, 2*SMSS) \qquad (3)$$

As discussed above, FlightSize is the amount of outstanding data in the network.

Implementation Note: an easy mistake to make is to simply use cwnd, rather than FlightSize, which in some implementations may incidentally increase well beyond rwnd.

Furthermore, upon a timeout cwnd MUST be set to no more than the loss window, LW, which equals 1 full-sized segment (regardless of the value of IW). Therefore, after retransmitting the dropped segment the TCP sender uses the slow start algorithm to increase the window from 1 full-sized segment to the new value of ssthresh, at which point congestion avoidance again takes over.

### 3.2 Fast Retransmit/Fast Recovery

A TCP receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives. The purpose of this ACK is to inform the sender that a segment was received out-of-order and which sequence number is expected. From the sender's perspective, duplicate ACKs can be caused by a number of network problems. First, they can be caused by dropped segments. In this case, all segments after the dropped segment will trigger duplicate ACKs. Second, duplicate ACKs can be caused by the re-ordering of data segments by the network (not a rare event along some network paths [Pax97]). Finally, duplicate ACKs can be caused by replication of ACK or data segments by the network. In addition, a TCP receiver SHOULD send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space. This will generate more timely information for a sender recovering from a loss through a retransmission timeout, a fast retransmit, or an experimental loss recovery algorithm, such as NewReno [FH98].

The TCP sender SHOULD use the "fast retransmit" algorithm to detect and repair loss, based on incoming duplicate ACKs. The fast retransmit algorithm uses the arrival of 3 duplicate ACKs (4 identical ACKs without the arrival of any other intervening packets) as an indication that a segment has been lost. After receiving 3 duplicate ACKs, TCP performs a retransmission of what appears to be the missing segment, without waiting for the retransmission timer to expire.

After the fast retransmit algorithm sends what appears to be the missing segment, the "fast recovery" algorithm governs the transmission of new data until a non-duplicate ACK arrives. The reason for not performing slow start is that the receipt of the duplicate ACKs not only indicates that a segment has been lost, but also that segments are most likely leaving the network (although a massive segment duplication by the network can invalidate this conclusion). In other words, since the receiver can only generate a duplicate ACK when a segment has arrived, that segment has left the network and is in the receiver's buffer, so we know it is no longer consuming network resources. Furthermore, since the ACK "clock" [Jac88] is preserved, the TCP sender can continue to transmit new segments (although transmission must continue using a reduced cwnd).

The fast retransmit and fast recovery algorithms are usually implemented together as follows.

1. When the third duplicate ACK is received, set ssthresh to no more than the value given in equation 3.

2. Retransmit the lost segment and set cwnd to ssthresh plus 3*SMSS. This artificially "inflates" the congestion window by the number of segments (three) that have left the network and which the receiver has buffered.

3. For each additional duplicate ACK received, increment cwnd by SMSS. This artificially inflates the congestion window in order to reflect the additional segment that has left the network.

4. Transmit a segment, if allowed by the new value of cwnd and the receiver's advertised window.

5. When the next ACK arrives that acknowledges new data, set cwnd to ssthresh (the value set in step 1). This is termed "deflating" the window.

   This ACK should be the acknowledgment elicited by the retransmission from step 1, one RTT after the retransmission (though it may arrive sooner in the presence of significant out-of-order delivery of data segments at the receiver). Additionally, this ACK should acknowledge all the intermediate segments sent between the lost segment and the receipt of the third duplicate ACK, if none of these were lost.

Note: This algorithm is known to generally not recover very efficiently from multiple losses in a single flight of packets [FF96]. One proposed set of modifications to address this problem can be found in [FH98].

## 4. Additional Considerations

### 4.1 Re-starting Idle Connections

A known problem with the TCP congestion control algorithms described above is that they allow a potentially inappropriate burst of traffic to be transmitted after TCP has been idle for a relatively long period of time. After an idle period, TCP cannot use the ACK clock to strobe new segments into the network, as all the ACKs have drained from the network. Therefore, as specified above, TCP can potentially send a cwnd-size line-rate burst into the network after an idle period.

[Jac88] recommends that a TCP use slow start to restart transmission after a relatively long idle period. Slow start serves to restart the ACK clock, just as it does at the beginning of a transfer. This mechanism has been widely deployed in the following manner. When TCP has not received a segment for more than one retransmission timeout, cwnd is reduced to the value of the restart window (RW) before

transmission begins.

For the purposes of this standard, we define RW = IW.

We note that the non-standard experimental extension to TCP defined in [AFP98] defines RW = min(IW, cwnd), with the definition of IW adjusted per equation (1) above.

Using the last time a segment was received to determine whether or not to decrease cwnd fails to deflate cwnd in the common case of persistent HTTP connections [HTH98]. In this case, a WWW server receives a request before transmitting data to the WWW browser. The reception of the request makes the test for an idle connection fail, and allows the TCP to begin transmission with a possibly inappropriately large cwnd.

Therefore, a TCP SHOULD set cwnd to no more than RW before beginning transmission if the TCP has not sent data in an interval exceeding the retransmission timeout.

### 4.2 Generating Acknowledgments

The delayed ACK algorithm specified in [Bra89] SHOULD be used by a TCP receiver. When used, a TCP receiver MUST NOT excessively delay acknowledgments. Specifically, an ACK SHOULD be generated for at least every second full-sized segment, and MUST be generated within 500 ms of the arrival of the first unacknowledged packet.

The requirement that an ACK "SHOULD" be generated for at least every second full-sized segment is listed in [Bra89] in one place as a SHOULD and another as a MUST. Here we unambiguously state it is a SHOULD. We also emphasize that this is a SHOULD, meaning that an implementor should indeed only deviate from this requirement after careful consideration of the implications. See the discussion of "Stretch ACK violation" in [PAD+98] and the references therein for a discussion of the possible performance problems with generating ACKs less frequently than every second full-sized segment.

In some cases, the sender and receiver may not agree on what constitutes a full-sized segment. An implementation is deemed to comply with this requirement if it sends at least one acknowledgment every time it receives 2*RMSS bytes of new data from the sender, where RMSS is the Maximum Segment Size specified by the receiver to the sender (or the default value of 536 bytes, per [Bra89], if the receiver does not specify an MSS option during connection establishment). The sender may be forced to use a segment size less than RMSS due to the maximum transmission unit (MTU), the path MTU discovery algorithm or other factors. For instance, consider the

case when the receiver announces an RMSS of X bytes but the sender ends up using a segment size of Y bytes (Y < X) due to path MTU discovery (or the sender's MTU size). The receiver will generate stretch ACKs if it waits for 2*X bytes to arrive before an ACK is sent. Clearly this will take more than 2 segments of size Y bytes. Therefore, while a specific algorithm is not defined, it is desirable for receivers to attempt to prevent this situation, for example by acknowledging at least every second segment, regardless of size. Finally, we repeat that an ACK MUST NOT be delayed for more than 500 ms waiting on a second full-sized segment to arrive.

Out-of-order data segments SHOULD be acknowledged immediately, in order to accelerate loss recovery. To trigger the fast retransmit algorithm, the receiver SHOULD send an immediate duplicate ACK when it receives a data segment above a gap in the sequence space. To provide feedback to senders recovering from losses, the receiver SHOULD send an immediate ACK when it receives a data segment that fills in all or part of a gap in the sequence space.

A TCP receiver MUST NOT generate more than one ACK for every incoming segment, other than to update the offered window as the receiving application consumes new data [page 42, Pos81][Cla82].

### 4.3 Loss Recovery Mechanisms

A number of loss recovery algorithms that augment fast retransmit and fast recovery have been suggested by TCP researchers. While some of these algorithms are based on the TCP selective acknowledgment (SACK) option [MMFR96], such as [FF96,MM96a,MM96b], others do not require SACKs [Hoe96,FF96,FH98]. The non-SACK algorithms use "partial acknowledgments" (ACKs which cover new data, but not all the data outstanding when loss was detected) to trigger retransmissions. While this document does not standardize any of the specific algorithms that may improve fast retransmit/fast recovery, these enhanced algorithms are implicitly allowed, as long as they follow the general principles of the basic four algorithms outlined above.

Therefore, when the first loss in a window of data is detected, ssthresh MUST be set to no more than the value given by equation (3).
Second, until all lost segments in the window of data in question are repaired, the number of segments transmitted in each RTT MUST be no more than half the number of outstanding segments when the loss was detected. Finally, after all loss in the given window of segments has been successfully retransmitted, cwnd MUST be set to no more than ssthresh and congestion avoidance MUST be used to further increase cwnd. Loss in two successive windows of data, or the loss of a retransmission, should be taken as two indications of congestion and, therefore, cwnd (and ssthresh) MUST be lowered twice in this case.

The algorithms outlined in [Hoe96,FF96,MM96a,MM6b] follow the principles of the basic four congestion control algorithms outlined in this document.

5. Security Considerations

This document requires a TCP to diminish its sending rate in the presence of retransmission timeouts and the arrival of duplicate acknowledgments. An attacker can therefore impair the performance of a TCP connection by either causing data packets or their acknowledgments to be lost, or by forging excessive duplicate acknowledgments. Causing two congestion control events back-to-back will often cut ssthresh to its minimum value of 2*SMSS, causing the connection to immediately enter the slower-performing congestion avoidance phase.

The Internet to a considerable degree relies on the correct implementation of these algorithms in order to preserve network stability and avoid congestion collapse. An attacker could cause TCP endpoints to respond more aggressively in the face of congestion by forging excessive duplicate acknowledgments or excessive acknowledgments for new data. Conceivably, such an attack could drive a portion of the network into congestion collapse.

6. Changes Relative to <A HREF="/rfcs/rfc2001.html">RFC 2001</A>

This document has been extensively rewritten editorially and it is not feasible to itemize the list of changes between the two documents. The intention of this document is not to change any of the recommendations given in <A HREF="/rfcs/rfc2001.html">RFC 2001</A>, but to further clarify cases that were not discussed in detail in 2001. Specifically, this document suggests what TCP connections should do after a relatively long idle period, as well as specifying and clarifying some of the issues

pertaining to TCP ACK generation. Finally, the allowable upper bound
for the initial congestion window has also been raised from one to
two segments.

Acknowledgments

The four algorithms that are described were developed by Van
Jacobson.

Some of the text from this document is taken from "TCP/IP
Illustrated, Volume 1: The Protocols" by W. Richard Stevens
(Addison-Wesley, 1994) and "TCP/IP Illustrated, Volume 2: The
Implementation" by Gary R. Wright and W. Richard Stevens (Addison-
Wesley, 1995). This material is used with the permission of
Addison-Wesley.

Neal Cardwell, Sally Floyd, Craig Partridge and Joe Touch contributed
a number of helpful suggestions.

References

[AFP98] Allman, M., Floyd, S. and C. Partridge, "Increasing TCP's
Initial Window Size, <A HREF="/rfcs/rfc2414.html">RFC
2414</A>, September 1998.

[Bra89] Braden, R., "Requirements for Internet Hosts --
Communication Layers", STD 3, <A
HREF="/rfcs/rfc1122.html">RFC 1122</A>, October 1989.

[Bra97] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, <A
HREF="/rfcs/rfc2119.html">RFC 2119</A>, March 1997.

[Cla82] Clark, D., "Window and Acknowledgment Strategy in TCP", RFC
813, July 1982.

[FF96] Fall, K. and S. Floyd, "Simulation-based Comparisons of
Tahoe, Reno and SACK TCP", Computer Communication Review,
July 1996. <A
HREF="ftp://ftp.ee.lbl.gov/papers/sacks.ps.Z">ftp://ftp.ee.lbl.gov/pap
ers/sacks.ps.Z</A>.

[FH98] Floyd, S. and T. Henderson, "The NewReno Modification to
TCP's Fast Recovery Algorithm", <A
HREF="/rfcs/rfc2582.html">RFC 2582</A>, April 1999.

[Flo94] Floyd, S., "TCP and Successive Fast Retransmits. Technical
report", October 1994.
<A
HREF="ftp://ftp.ee.lbl.gov/papers/fastretrans.ps">ftp://ftp.ee.lbl.gov
/papers/fastretrans.ps</A>.

[Hoe96] Hoe, J., "Improving the Start-up Behavior of a Congestion
Control Scheme for TCP", In ACM SIGCOMM, August 1996.

[HTH98] Hughes, A., Touch, J. and J. Heidemann, "Issues in TCP
Slow-Start Restart After Idle", Work in Progress.

[Jac88] Jacobson, V., "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988. <A HREF="ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z">ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z</A>.

[Jac90] Jacobson, V., "Modified TCP Congestion Avoidance Algorithm", end2end-interest mailing list, April 30, 1990. <A HREF="ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail">ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail</A>.

[MD90] Mogul, J. and S. Deering, "Path MTU Discovery", <A HREF="/rfcs/rfc1191.html">RFC 1191</A>, November 1990.

[MM96a] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", Proceedings of SIGCOMM'96, August, 1996, Stanford, CA. Available from<A HREF="http://www.psc.edu/networking/papers/papers.html">http://www.psc.edu/networking/papers/papers.html</A>

[MM96b] Mathis, M. and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Technical report. Available from <A HREF="http://www.psc.edu/networking/papers/FACKnotes/current">http://www.psc.edu/networking/papers/FACKnotes/current</A>.

[MMFR96] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, "TCP Selective Acknowledgement Options", <A HREF="/rfcs/rfc2018.html">RFC 2018</A>, October 1996.

[PAD+98] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J. and B. Volz, "Known TCP Implementation Problems", <A HREF="/rfcs/rfc2525.html">RFC 2525</A>, March 1999.

[Pax97] Paxson, V., "End-to-End Internet Packet Dynamics", Proceedings of SIGCOMM '97, Cannes, France, Sep. 1997.

[Pos81] Postel, J., "Transmission Control Protocol", STD 7, <A HREF="/rfcs/rfc793.html">RFC 793</A>, September 1981.

[Ste94] Stevens, W., "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley, 1994.

[Ste97] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", <A HREF="/rfcs/rfc2001.html">RFC 2001</A>, January 1997.

[WS95] Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1995.

Authors' Addresses

Mark Allman
NASA Glenn Research Center/Sterling Software
Lewis Field
21000 Brookpark Rd.  MS 54-2
Cleveland, OH  44135
216-433-6586

EMail: <A
HREF="mailto:mallman@grc.nasa.gov">mallman@grc.nasa.gov</A>
<A
HREF="http://roland.grc.nasa.gov/~mallman">http://roland.grc.nasa.gov/
~mallman</A>

Vern Paxson
ACIRI / ICSI
1947 Center Street
Suite 600
Berkeley, CA 94704-1198

Phone: +1 510/642-4274 x302
EMail: <A HREF="mailto:vern@aciri.org">vern@aciri.org</A>

W. Richard Stevens
1202 E. Paseo del Zorro
Tucson, AZ  85718
520-297-9416

## Full Copyright Statement