

## Authentication, Authorization and Accounting (AAA) Transport Profile

### 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

Copyright (C) The Internet Society (2003). All Rights Reserved.

### Abstract

This document discusses transport issues that arise within protocols for Authentication, Authorization and Accounting (AAA). It also provides recommendations on the use of transport by AAA protocols. This includes usage of standards-track RFCs as well as experimental proposals.

### Table of Contents

1.	Introduction . . . . .	2
1.1.	Requirements Language. . . . .	2
1.2.	Terminology. . . . .	2
2.	Issues in Transport Usage. . . . .	5
2.1.	Application-driven Versus Network-driven . . . . .	5
2.2.	Slow Failover. . . . .	6
2.3.	Use of Nagle Algorithm . . . . .	7
2.4.	Multiple Connections . . . . .	7
2.5.	Duplicate Detection. . . . .	8
2.6.	Invalidation of Transport Parameter Estimates. . . . .	8
2.7.	Inability to use Fast Re-Transmit. . . . .	9
2.8.	Congestion Avoidance . . . . .	9
2.9.	Delayed Acknowledgments. . . . .	11
2.10.	Premature Failover . . . . .	11
2.11.	Head of Line Blocking. . . . .	11
2.12.	Connection Load Balancing. . . . .	12

3.	AAA Transport Profile. . . . .	12
3.1.	Transport Mappings . . . . .	12
3.2.	Use of Nagle Algorithm . . . . .	12
3.3.	Multiple Connections . . . . .	13
3.4.	Application Layer Watchdog . . . . .	13
3.5.	Duplicate Detection. . . . .	19
3.6.	Invalidation of Transport Parameter Estimates. . . . .	20
3.7.	Inability to use Fast Re-Transmit. . . . .	21
3.8.	Head of Line Blocking. . . . .	22
3.9.	Congestion Avoidance . . . . .	23
3.10.	Premature Failover . . . . .	24
4.	Security Considerations. . . . .	24
5.	IANA Considerations. . . . .	25
6.	References . . . . .	25
6.1.	Normative References . . . . .	25
6.2.	Informative References . . . . .	26
	Appendix A - Detailed Watchdog Algorithm Description . . . . .	28
	Appendix B - AAA Agents. . . . .	33
	B.1. Relays and Proxies . . . . .	33
	B.2. Re-directs . . . . .	35
	B.3. Store and Forward Proxies. . . . .	36
	B.4. Transport Layer Proxies. . . . .	38
	Intellectual Property Statement. . . . .	39
	Acknowledgments. . . . .	39
	Author Addresses . . . . .	40
	Full Copyright Statement . . . . .	41

## 1. Introduction

This document discusses transport issues that arise within protocols for Authentication, Authorization and Accounting (AAA). It also provides recommendations on the use of transport by AAA protocols. This includes usage of standards-track RFCs as well as experimental proposals.

### 1.1. Requirements Language

In this document, the key words "MAY", "MUST", "MUST NOT", "optional", "recommended", "SHOULD", and "SHOULD NOT", are to be interpreted as described in [RFC2119].

### 1.2. Terminology

#### Accounting

The act of collecting information on resource usage for the purpose of trend analysis, auditing, billing, or cost allocation.

**Administrative Domain**

An internet, or a collection of networks, computers, and databases under a common administration.

**Agent**

A AAA agent is an intermediary that communicates with AAA clients and servers. Several types of AAA agents exist, including Relays, Re-directs, and Proxies.

**Application-driven transport**

Transport behavior is said to be "application-driven" when the rate at which messages are sent is limited by the rate at which the application generates data, rather than by the size of the congestion window. In the most extreme case, the time between transactions exceeds the round-trip time between sender and receiver, implying that the application operates with an effective congestion window of one. AAA transport is typically application driven.

**Attribute Value Pair (AVP)**

The variable length concatenation of a unique Attribute (represented by an integer) and a Value containing the actual value identified by the attribute.

**Authentication**

The act of verifying a claimed identity, in the form of a pre-existing label from a mutually known name space, as the originator of a message (message authentication) or as the end-point of a channel (entity authentication).

**Authorization**

The act of determining if a particular right, such as access to some resource, can be granted to the presenter of a particular credential.

**Billing** The act of preparing an invoice.

**Network Access Identifier**

The Network Access Identifier (NAI) is the userID submitted by the host during network access authentication. In roaming, the purpose of the NAI is to identify the user as well as to assist in the routing of the authentication request. The NAI may not necessarily be the same as the user's e-mail address or the user-ID submitted in an application layer authentication.

#### Network Access Server (NAS)

A Network Access Server (NAS) is a device that hosts connect to in order to get access to the network.

#### Proxy

In addition to forwarding requests and responses, proxies enforce policies relating to resource usage and provisioning. This is typically accomplished by tracking the state of NAS devices. While proxies typically do not respond to client Requests prior to receiving a Response from the server, they may originate Reject messages in cases where policies are violated. As a result, proxies need to understand the semantics of the messages passing through them, and may not support all extensions.

#### Local Proxy

A Local Proxy is a proxy that exists within the same administrative domain as the network device (e.g. NAS) that issued the AAA request. Typically a local proxy is used to multiplex AAA messages to and from a large number of network devices, and may implement policy.

#### Store and forward proxy

Store and forward proxies distinguish themselves from other proxy species by sending a reply to the NAS prior to proxying the request to the server. As a result, store and forward proxies need to implement AAA client and server functionality for the messages that they handle. Store and Forward proxies also typically keep state on conversations in progress in order to assure delivery of proxied Requests and Responses. While store and forward proxies are most frequently deployed for accounting, they also can be used to implement authentication/authorization policy.

#### Network-driven transport

Transport behavior is said to be "network driven" when the rate at which messages are sent is limited by the congestion window, not by the rate at which the application can generate data. File transfer is an example of an application where transport is network driven.

#### Re-direct

Rather than forwarding Requests and Responses between clients and servers, Re-directs refer clients to servers and allow them to communicate directly. Since Re-directs do not sit in the forwarding path, they do not alter any AVPs transitting between client and server. Re-directs do not originate messages and are capable of handling any message type. A Re-direct may be configured only to re-direct messages of certain types, while acting as a Relay

or Proxy for other types. As with Relays, re-directs do not keep state with respect to conversations or NAS resources.

Relay Relays forward requests and responses based on routing-related AVPs and domain forwarding table entries. Since relays do not enforce policies, they do not examine or alter non-routing AVPs. As a result, relays never originate messages, do not need to understand the semantics of messages or non-routing AVPs, and are capable of handling any extension or message type. Since relays make decisions based on information in routing AVPs and domain forwarding tables they do not keep state on NAS resource usage or conversations in progress.

## 2. Issues in AAA Transport Usage

Issues that arise in AAA transport usage include:

- Application-driven versus network-driven
- Slow failover
- Use of Nagle Algorithm
- Multiple connections
- Duplicate detection
- Invalidation of transport parameter estimates
- Inability to use fast re-transmit
- Congestion avoidance
- Delayed acknowledgments
- Premature Failover
- Head of line blocking
- Connection load balancing

We discuss each of these issues in turn.

### 2.1. Application-driven versus Network-driven

AAA transport behavior is typically application rather than network driven. This means that the rate at which messages are sent is typically limited by how quickly they are generated by the application, rather than by the size of the congestion window.

For example, let us assume a 48-port NAS with an average session time of 20 minutes. This device will, on average, send only 144 authentication/authorization requests/hour, and an equivalent number of accounting requests. This represents an average inter-packet spacing of 25 seconds, which is much larger than the Round Trip Time (RTT) in most networks.

Even on much larger NAS devices, the inter-packet spacing is often larger than the RTT. For example, consider a 2048-port NAS with an average session time of 10 minutes. It will on average send 3.4 authentication/authorization requests/second, and an equivalent number of accounting requests. This translates to an average inter-packet spacing of 293 ms.

However, even where transport behavior is largely application-driven, periods of network-driven behavior can occur. For example, after a NAS reboot, previously stored accounting records may be sent to the accounting server in rapid succession. Similarly, after recovery from a power failure, users may respond with a large number of simultaneous logins. In both cases, AAA messages may be generated more quickly than the network will allow them to be sent, and a queue will build up.

Network congestion can occur when transport behavior is network-driven or application-driven. For example, while a single NAS may not send substantial AAA traffic, many NASes may communicate with a single AAA proxy or server. As a result, routers close to a heavily loaded proxy or server may experience congestion, even though traffic from each individual NAS is light. Such "convergent congestion" can result in dropped packets in routers near the AAA server, or even within the AAA server itself.

Let us consider what happens when 10,000 48-ports NASes, each with an average session time of 20 minutes, are configured with the same AAA agent or server. The unfortunate proxy or server would receive 400 authentication/authorization requests/second and an equivalent number of accounting requests. For 1000 octet requests, this would generate 6.4 Mbps of incoming traffic at the AAA agent or server.

While this transaction load is within the capabilities of the fastest AAA agents and servers, implementations exist that cannot handle such a high load. Thus high queuing delays and/or dropped packets may be experienced at the agent or server, even if routers on the path are not congested. Thus, a well designed AAA protocol needs to be able to handle congestion occurring at the AAA server, as well as congestion experienced within the network.

## 2.2. Slow Failover

Where TCP [RFC793] is used as the transport, AAA implementations will experience very slow fail over times if they wait until a TCP connection times out before resending on another connection. This is not an issue for SCTP [RFC2960], which supports endpoint and path failure detection. As described in section 8 of [RFC2960], when the number of retransmissions exceeds the maximum

("Association.Max.Retrans"), the peer endpoint is considered unreachable, the association enters the CLOSED state, and the failure is reported to the application. This enables more rapid failure detection.

### 2.3. Use of Nagle Algorithm

AAA protocol messages are often smaller than the maximum segment size (MSS). While exceptions occur when certificate-based authentication messages are issued or where a low path MTU is found, typically AAA protocol messages are less than 1000 octets. Therefore, when using TCP [RFC793], the total packet count and associated network overhead can be reduced by combining multiple AAA messages within a single packet.

Where AAA runs over TCP and transport behavior is network-driven, such as after a reboot when many users login simultaneously, or many stored accounting records need to be sent, the Nagle algorithm will result in "transport layer batching" of AAA messages. While this does not reduce the work required by the application in parsing packets and responding to the messages, it does reduce the number of packets processed by routers along the path. The Nagle algorithm is not used with SCTP.

Where AAA transport is application-driven, the NAS will typically receive a reply from the home server prior to having another request to send. This implies, for example, that accounting requests will typically be sent individually rather than being batched by the transport layer. As a result, within the application-driven regime, the Nagle algorithm [RFC896] is ineffective.

### 2.4. Multiple Connections

Since the RADIUS [RFC2865] Identifier field is a single octet, a maximum of 256 requests can be in progress between two endpoints described by a 5-tuple: (Client IP address, Client port, UDP, Server IP address, Server port). In order to get around this limitation, RADIUS clients have utilized more than one sending port, sometimes even going to the extreme of using a different UDP source port for each NAS port.

Were this behavior to be extended to AAA protocols operating over reliable transport, the result would be multiplication of the effective slow-start ramp-up by the number of connections. For example, if a AAA client had ten connections open to a AAA agent, and used a per-connection initial window [RFC3390] of 2, then the

effective initial window would be 20. This is inappropriate, since it would permit the AAA client to send a large burst of packets into the network.

## 2.5. Duplicate Detection

Where a AAA client maintains connections to multiple AAA agents or servers, and where failover/failback or connection load balancing is supported, it is possible for multiple agents or servers to receive duplicate copies of the same transaction. A transaction may be sent on another connection before expiration of the "time wait" interval necessary to guarantee that all packets sent on the original connection have left the network. Therefore it is conceivable that transactions sent on the alternate connection will arrive before those sent on the failed connection. As a result, AAA agents and servers **MUST** be prepared to handle duplicates, and **MUST** assume that duplicates can arrive on any connection.

For example, in billing, it is necessary to be able to weed out duplicate accounting records, based on the accounting session-id, event-timestamp and NAS identification information. Where authentication requests are always idempotent, the resultant duplicate responses from multiple servers will presumably be identical, so that little harm will result.

However, there are situations where the response to an authentication request will depend on a previously established state, such as when simultaneous usage restrictions are being enforced. In such cases, authentication requests will not be idempotent. For example, while an initial request might elicit an Accept response, a duplicate request might elicit a Reject response from another server, if the user were already presumed to be logged in, and only one simultaneous session were permitted. In these situations, the AAA client might receive both Accept and Reject responses to the same duplicate request, and the outcome will depend on which response arrives first.

## 2.6. Invalidation of Transport Parameter Estimates

Congestion control principles [Congest],[RFC2914] require the ability of a transport protocol to respond effectively to congestion, as sensed via increasing delays, packet loss, or explicit congestion notification.

With network-driven applications, it is possible to respond to congestion on a timescale comparable to the round-trip time (RTT).

However, with AAA protocols, the time between sends may be longer than the RTT, so that the network conditions can not be assumed to



persist between sends. For example, the congestion window may grow during a period in which congestion is being experienced because few packets are sent, limiting the opportunity for feedback. Similarly, after congestion is detected, the congestion window may remain small, even though the network conditions that existed at the time of congestion no longer apply by the time when the next packets are sent. In addition, due to the low sampling interval, estimates of RTT and RTO made via the procedure described in [RFC2988] may become invalid.

## 2.7. Inability to Use Fast Re-transmit

When congestion window validation [RFC2861] is implemented, the result is that AAA protocols operate much of the time in slow-start with an initial congestion window set to 1 or 2, depending on the implementation [RFC3390]. This implies that AAA protocols gain little benefit from the windowing features of reliable transport.

Since the congestion window is so small, it is generally not possible to receive enough duplicate ACKs (3) to trigger fast re-transmit. In addition, since AAA traffic is two-way, ACKs including data will not count as part of the duplicate ACKs necessary to trigger fast re-transmit. As a result, dropped packets will require a retransmission timeout (RTO).

## 2.8. Congestion Avoidance

The law of conservation of packets [Congest] suggests that a client should not send another packet into the network until it can be reasonably sure that a packet has exited the network on the same path. In the case of a AAA client, the law suggests that it should not retransmit to the same server or choose another server until it can be reasonably sure that a packet has exited the network on the same path. If the client advances the window as responses arrive, then the client will "self clock", adjusting its transmission rate to the available bandwidth.

While a AAA client using a reliable transport such as TCP [RFC793] or SCTP [RFC2960] will self-clock when communicating directly with a AAA-server, end-to-end self-clocking is not assured when AAA agents are present.

As described in the Appendix, AAA agents include Relays, Proxies, Re-directs, Store and Forward proxies, and Transport proxies. Of these agents, only Transport proxies and Re-directs provide a direct transport connection between the AAA client and server, allowing end-to-end self-clocking to occur.

With Relays, Proxies or Store and Forward proxies, two separate and de-coupled transport connections are used. One connection operates between the AAA client and agent, and another between the agent and server. Since the two transport connections are de-coupled, transport layer ACKs do not flow end-to-end, and self-clocking does not occur.

For example, consider what happens when the bottleneck exists between a AAA Relay and a AAA server. Self-clocking will occur between the AAA client and AAA Relay, causing the AAA client to adjust its sending rate to the rate at which transport ACKs flow back from the AAA Relay. However, since this rate is higher than the bottleneck bandwidth, the overall system will not self-clock.

Since there is no direct transport connection between the AAA client and AAA server, the AAA client does not have the ability to estimate end-to-end transport parameters and adjust its sending rate to the bottleneck bandwidth between the Relay and server. As a result, the incoming rate at the AAA Relay can be higher than the rate at which packets can be sent to the AAA server.

In this case, the end-to-end performance will be determined by details of the agent implementation. In general, the end-to-end transport performance in the presence of Relays, Proxies or Store and Forward proxies will always be worse in terms of delay and packet loss than if the AAA client and server were communicating directly.

For example, if the agent operates with a large receive buffer, it is possible that a large queue will develop on the receiving side, since the AAA client is able to send packets to the AAA agent more rapidly than the agent can send them to the AAA server. Eventually, the buffer will overflow, causing wholesale packet loss as well as high delay.

Methods to induce fine-grained coupling between the two transport connections are difficult to implement. One possible solution is for the AAA agent to operate with a receive buffer that is no larger than its send buffer. If this is done, "back pressure" (closing of the receive window) will cause the agent to reduce the AAA client sending rate when the agent send buffer fills. However, unless multiple connections exist between the AAA client and AAA agent, closing of the receive window will affect all traffic sent by the AAA client, even traffic destined to AAA servers where no bottleneck exists. Since multiple connections between a AAA client and agent result in multiplication of the effective slow-start ramp rate, this is not recommended. As a result, use of "back pressure" cannot enable individual AAA client-server conversations to self-clock, and this technique appears impractical for use in AAA.

## 2.9. Delayed Acknowledgments

As described in Appendix B, ACKs may comprise as much as half of the traffic generated in a AAA exchange. This occurs because AAA conversations are typically application-driven, and therefore there is frequently not enough traffic to enable ACK piggybacking. As a result, AAA protocols running over TCP or SCTP transport may experience a doubling of traffic as compared with implementations utilizing UDP transport.

It is typically not possible to address this issue via the sockets API. ACK parameters (such as the value of the delayed ACK timer) are typically fixed by TCP and SCTP implementations and are therefore not tunable by the application.

## 2.10. Premature Failover

RADIUS failover implementations are typically based on the concept of primary and secondary servers, in which all traffic flows to the primary server unless it is unavailable. However, the failover algorithm was not specified in [RFC2865] or [RFC2866]. As a result, RADIUS failover implementations vary in quality, with some failing over prematurely, violating the law of "conservation of packets".

Where a Relay, Proxy or Store and Forward proxy is present, the AAA client has no direct connection to a AAA server, and is unable to estimate the end-to-end transport parameters. As a result, a AAA client awaiting an application-layer response from the server has no transport-based mechanism for determining an appropriate failover timer.

For example, if the path between the AAA agent and server includes a high delay link, or if the AAA server is very heavily loaded, it is possible that the NAS will failover to another agent while packets are still in flight. This violates the principle of "conservation of packets", since the AAA client will inject additional packets into the network before having evidence that a previously sent packet has left the network. Such behavior can result in a worse situation on an already congested link, resulting in congestive collapse [Congest].

## 2.11. Head of Line Blocking

Head of line blocking occurs during periods of packet loss where the time between sends is shorter than the re-transmission timeout value (RTO). In such situations, packets back up in the send queue until

the lost packet can be successfully re-transmitted. This can be an issue for SCTP when using ordered delivery over a single stream, and for TCP.

Head of line blocking is typically an issue only on larger NASes. For example, a 48-port NAS with an average inter-packet spacing of 25 seconds is unlikely to have an RTO greater than this, unless severe packet loss has been experienced. However, a 2048-port NAS with an average inter-packet spacing of 293 ms may experience head-of-line blocking since the inter-packet spacing is less than the minimum RTO value of 1 second [RFC2988].

## 2.12. Connection Load Balancing

In order to lessen queuing delays and address head of line blocking, a AAA implementation may wish to load balance between connections to multiple destinations. While it is possible to employ dynamic load balancing techniques, this level of sophistication may not be required. In many situations, adequate reliability and load balancing can be achieved via static load balancing, where traffic is distributed between destinations based on static "weights".

## 3. AAA Transport Profile

In order to address AAA transport issues, it is recommended that AAA protocols make use of standards track as well as experimental techniques. More details are provided in the sections that follow.

### 3.1. Transport Mappings

AAA Servers **MUST** support TCP and SCTP. AAA clients **SHOULD** support SCTP, but **MUST** support TCP if SCTP is not available. As support for SCTP improves, it is possible that SCTP support will be required on clients at some point in the future. AAA agents inherit all the obligations of Servers with respect to transport support.

### 3.2. Use of Nagle Algorithm

While AAA protocols typically operate in the application-driven regime, there are circumstances in which they are network driven. For example, where an NAS reboots, or where connectivity is restored between an NAS and a AAA agent, it is possible that multiple packets will be available for sending.

As a result, there are circumstances where the transport-layer batching provided by the Nagle Algorithm (12) is useful, and as a result, AAA implementations running over TCP **MUST** enable the Nagle algorithm, [RFC896]. The Nagle algorithm is not used with SCTP.

### 3.3. Multiple Connections

AAA protocols SHOULD use only a single persistent connection between a AAA client and a AAA agent or server. They SHOULD provide for pipelining of requests, so that more than one request can be in progress at a time. In order to minimize use of inactive connections in roaming situations, a AAA client or agent MAY bring down a connection to a AAA agent or server if the connection has been unutilized (discounting the watchdog) for a certain period of time, which MUST NOT be less than BRINGDOWN\_INTERVAL (5 minutes).

While a AAA client/agent SHOULD only use a single persistent connection to a given AAA agent or server, it MAY have connections to multiple AAA agents or servers. A AAA client/agent connected to multiple agents/servers can treat them as primary/secondary or balance load between them.

### 3.4. Application Layer Watchdog

In order to enable AAA implementations to more quickly detect transport and application-layer failures, AAA protocols MUST support an application layer watchdog message.

The application layer watchdog message enables failover from a peer that has failed, either because it is unreachable or because its applications functions have failed. This is distinct from the purpose of the SCTP heartbeat, which is to enable failover between interfaces. The SCTP heartbeat may enable a failover to another path to reach the same server, but does not address the situation where the server system or the application service has failed. Therefore both mechanisms MAY be used together.

The watchdog is used in order to enable a AAA client or agent to determine when to resend on another connection. It operates on all open connections and is used to suspend and eventually close connections that are experiencing difficulties. The watchdog is also used to re-open and validate connections that have returned to health. The watchdog may be utilized either within primary/secondary or load balancing configurations. However, it is not intended as a cluster heartbeat mechanism.

The application layer watchdog is designed to detect failures of the immediate peer, and not to be affected by failures of downstream proxies or servers. This prevents instability in downstream AAA components from propagating upstream. While the receipt of any AAA Response from a peer is taken as evidence that the peer is up, lack of a Response is insufficient to conclude that the peer is down. Since the lack of Response may be the result of problems with a

downstream proxy or server, only after failure to respond to the watchdog message can it be determined that the peer is down.

Since the watchdog algorithm takes any AAA Response into account in determining peer liveness, decreases in the watchdog timer interval do not significantly increase the level of watchdog traffic on heavily loaded networks. This is because watchdog messages do not need to be sent where other AAA Response traffic serves as a constant reminder of peer liveness. Watchdog traffic only increases when AAA traffic is light, and therefore a AAA Response "signal" is not present. Nevertheless, decreasing the timer interval TWINIT does increase the probability of false failover significantly, and so this decision should be made with care.

#### 3.4.1. Algorithm Overview

The watchdog behavior is controlled by an algorithm defined in this section. This algorithm is appropriate for use either within primary/secondary or load balancing configurations. Implementations SHOULD implement this algorithm, which operates as follows:

- [1] Watchdog behavior is controlled by a single timer ( $T_w$ ). The initial value of  $T_w$ , prior to jittering is  $T_{winit}$ . The default value of  $T_{winit}$  is 30 seconds. This value was selected because it minimizes the probability that failover will be initiated due to a routing flap, as noted in [Paxson].

While  $T_{winit}$  MAY be set as low as 6 seconds (not including jitter), it MUST NOT be set lower than this. Note that setting such a low value for  $T_{winit}$  is likely to result in an increased probability of duplicates, as well as an increase in spurious failover and fallback attempts.

In order to avoid synchronization behaviors that can occur with fixed timers among distributed systems, each time the watchdog interval is calculated with a jitter by using the  $T_{winit}$  value and randomly adding a value drawn between -2 and 2 seconds. Alternative calculations to create jitter MAY be used. These MUST be pseudo-random, generated by a PRNG seeded as per [RFC1750].

- [2] When any AAA message is received,  $T_w$  is reset. This need not be a response to a watchdog request. Receiving a watchdog response from a peer constitutes activity, and  $T_w$  should be reset. If the watchdog timer expires and no watchdog response is pending, then a watchdog message is sent. On sending a watchdog request,  $T_w$  is reset.

Watchdog packets are not retransmitted by the AAA protocol, since AAA protocols run over reliable transports that will handle all retransmissions internally. As a result, a watchdog request is only sent when there is no watchdog response pending.

- [3] If the watchdog timer expires and a watchdog response is pending, then failover is initiated. In order for a AAA client or agent to perform failover procedures, it is necessary to maintain a pending message queue for a given peer. When an answer message is received, the corresponding request is removed from the queue. The Hop-by-Hop Identifier field MAY be used to match the answer with the queued request.

When failover is initiated, all messages in the queue are sent to an alternate agent, if available. Multiple identical requests or answers may be received as a result of a failover. The combination of an end-to-end identifier and the origin host MUST be used to identify duplicate messages.

Note that where traffic is heavy, the application layer watchdog can take as long as 2Tw to determine that a peer has gone down. For peers receiving a high volume of AAA Requests, AAA Responses will continually reset the timer, so that after a failure it will take Tw for the lack of traffic to be noticed, and for the watchdog message to be sent. Another Tw will elapse before failover is initiated.

On a lightly loaded network without much AAA Response traffic, the watchdog timer will typically expire without being reset, so that a watchdog response will be outstanding and failover will be initiated after only a single timer interval has expired.

- [4] The client MUST NOT close the primary connection until the primary's watchdog timer has expired at least twice without a response (note that the watchdog is not sent a second time, however). Once this has occurred, the client SHOULD cause a transport reset or close to be done on the connection.

Once the primary connection has failed, subsequent requests are sent to the alternate server until the watchdog timer on the primary connection is reset.

Suspension of the primary connection prevents flapping between primary and alternate connections, and ensures that failover behavior remains consistent. The application may not receive a response to the watchdog request message due to a connectivity problem, in which case a transport layer ACK will not have been received, or the lack of response may be due to an application

problem. Without transport layer visibility, the application is unable to tell the difference, and must behave conservatively.

In situations where no transport layer ACK is received on the primary connection after multiple re-transmissions, the RTO will be exponentially backed off as described in [RFC2988]. Due to Karn's algorithm as implemented in SCTP and TCP, the RTO estimator will not be reset until another ACK is received in response to a non-re-transmitted request. Thus, in cases where the problem occurs at the transport layer, after the client fails over to the alternate server, the RTO of the primary will remain at a high value unless an ACK is received on the primary connection.

In the case where the problem occurs at the transport layer, subsequent requests sent on the primary connection will not receive the same service as was originally provided. For example, instead of failover occurring after 3 retransmissions, failover might occur without even a single retransmission if RTO has been sufficiently backed off. Of course, if the lack of a watchdog response was due to an application layer problem, then RTO will not have been backed off. However, without transport layer visibility, there is no way for the application to know this.

Suspending use of the primary connection until a response to a watchdog message is received guarantees that the RTO timer will have been reset before the primary connection is reused. If no response is received after the second watchdog timer expiration, then the primary connection is closed and the suspension becomes permanent.

- [5] While the connection is in the closed state, the AAA client MUST NOT attempt to send further watchdog messages on the connection. However, after the connection is closed, the AAA client continues to periodically attempt to reopen the connection.

The AAA client SHOULD wait for the transport layer to report connection failure before attempting again, but MAY choose to bound this wait time by the watchdog interval, Tw. If the connection is successfully opened, then the watchdog message is sent. Once three watchdog messages have been sent and responded to, the connection is returned to service, and transactions are once again sent over it. Connection validation via receipt of multiple watchdogs is not required when a connection is initially brought up -- in this case, the connection can immediately be put into service.



- [6] When using SCTP as a transport, it is not necessary to disable SCTP's transport-layer heartbeats. However, if AAA implementations have access to SCTP's heartbeat parameters, they MAY chose to ensure that SCTP's heartbeat interval is longer than the AAA watchdog interval, Tw. This will ensure that alternate paths are still probed by SCTP, while the primary path has a minimum of heartbeat redundancy.

#### 3.4.2. Primary/Secondary Failover Support

The watchdog timer MAY be integrated with primary/secondary style failover so as to provide improved reliability and basic load balancing. In order to balance load among multiple AAA servers, each AAA server is designated the primary for a portion of the clients, and designated as secondaries of varying priority for the remainder. In this way, load can be balanced among the AAA servers.

Within primary/secondary configurations, the watchdog timer operates as follows:

- [1] Assume that each client or agent is initially configured with a single primary agent or server, and one or more secondary connections.
- [2] The watchdog mechanism is used to suspend and eventually close primary connections that are experiencing difficulties. It is also used to re-open and validate connections that have returned to health.
- [3] Once a secondary is promoted to primary status, either on a temporary or permanent basis, the next server on the list of secondaries is promoted to fill the open secondary slot.
- [4] The client or agent periodically attempts to re-open closed connections, so that it is possible that a previously closed connection can be returned to service and become eligible for use again. Implementations will typically retain a limit on the number of connections open at a time, so that once a previously closed connection is brought online again, the lowest priority secondary connection will be closed. In order to prevent periodic closing and re-opening of secondary connections, it is recommended that functioning connections remain open for a minimum of 5 minutes.
- [5] In order to enable diagnosis of failover behavior, it is recommended that a table of failover events be kept within the MIB. These failover events SHOULD include appropriate transaction identifiers so that client and server data can be

compared, providing insight into the cause of the problem (transport or application layer).

### 3.4.3. Connection Load Balancing

Primary/secondary failover is capable of providing improved resilience and basic load balancing. However, it does not address TCP head of line blocking, since only a single connection is in use at a time.

A AAA client or agent maintaining connections to multiple agents or servers MAY load balance between them. Establishing connections to multiple agents or servers reduces, but does not eliminate, head of line blocking issues experienced on TCP connections. This issue does not exist with SCTP connections utilizing multiple streams.

In connection load balancing configurations, the application watchdog operates as follows:

- [1] Assume that each client or agent is initially configured with connections to multiple AAA agents or servers, with one connection between a given client/agent and an agent/server.
- [2] In static load balancing, transactions are apportioned among the connections based on the total number of connections and a "weight" assigned to each connection. Pearson's hash [RFC3074] applied to the NAI [RFC2486] can be used to determine which connection will handle a given transaction. Hashing on the NAI provides highly granular load balancing, while ensuring that all traffic for a given conversation will be sent to the same agent or server. In dynamic load balancing, the value of the "weight" can vary based on conditions such as AAA server load. Such techniques, while sophisticated, are beyond the scope of this document.
- [3] Transactions are distributed to connections based on the total number of available connections and their weights. A change in the number of available connections forces recomputation of the hash table. In order not to cause conversations in progress to be switched to new destinations, on recomputation, a transitional period is required in which both old and new hash tables are needed in order to permit aging out of conversations in progress. Note that this requires a way to easily determine whether a Request represents a new conversation or the continuation of an existing conversation. As a result, removing and adding of connections is an expensive operation, and it is recommended that the hash table only be recomputed once a connection is closed or returned to service.

Suspended connections, although they are not used, do not force hash table reconfiguration until they are closed. Similarly, re-opened connections not accumulating sufficient watchdog responses do not force a reconfiguration until they are returned to service.

While a connection is suspended, transactions that were to have been assigned to it are instead assigned to the next available server. While this results in a momentary imbalance, it is felt that this is a relatively small price to pay in order to reduce hash table thrashing.

- [4] In order to enable diagnosis of load balancing behavior, it is recommended that in addition to a table of failover events, a table of statistics be kept on each client, indexed by a AAA server. That way, the effectiveness of the load balancing algorithm can be evaluated.

### 3.5. Duplicate Detection

Multiple facilities are required to enable duplicate detection. These include session identifiers as well as hop-by-hop and end-to-end message identifiers. Hop-by-hop identifiers whose value may change at each hop are not sufficient, since a AAA server may receive the same message from multiple agents. For example, a AAA client can send a request to Agent1, then failover and resend the request to Agent2; both agents forward the request to the home AAA server, with different hop-by-hop identifiers. A Session Identifier is insufficient as it does not distinguish different messages for the the same session.

Proper treatment of the end-to-end message identifier ensures that AAA operations are idempotent. For example, without an end-to-end identifier, a AAA server keeping track of simultaneous logins might send an Accept in response to an initial Request, and then a Reject in response to a duplicate Request (where the user was allowed only one simultaneous login). Depending on which Response arrived first, the user might be allowed access or not.

However, if the server were to store the end-to-end message identifier along with the simultaneous login information, then the duplicate Request (which utilizes the same end-to-end message identifier) could be identified and the correct response could be returned.

### 3.6. Invalidation of Transport Parameter Estimates

In order to address invalidation of transport parameter estimates, AAA protocol implementations MAY utilize Congestion Window Validation [RFC2861] and RTO validation when using TCP. This specification also recommends a procedure for RTO validation.

[RFC2581] and [RFC2861] both recommend that a connection go into slow-start after a period where no traffic has been sent within the RTO interval. [RFC2861] recommends only increasing the congestion window if it was full when the ACK arrived. The congestion window is reduced by half once every RTO interval if no traffic is received.

When Congestion Window Validation is used, the congestion window will not build during application-driven periods, and instead will be decayed. As a result, AAA applications operating within the application-driven regime will typically run with a congestion window equal to the initial window much of the time, operating in "perpetual slowstart".

During periods in which AAA behavior is application-driven this will have no effect. Since the time between packets will be larger than RTT, AAA will operate with an effective congestion window equal to the initial window. However, during network-driven periods, the effect will be to space out sending of AAA packets. Thus instead of being able to send a large burst of packets into the network, a client will need to wait several RTTs as the congestion window builds during slow-start.

For example, a client operating over TCP with an initial window of 2, with 35 AAA requests to send would take approximately 6 RTTs to send them, as the congestion window builds during slow start: 2, 3, 3, 6, 9, 12. After the backlog is cleared, the implementation will once again be application-driven and the congestion window size will decay. If the client were using SCTP, the number of RTTs needed to transmit all requests would usually be less, and would depend on the size of the requests, since SCTP tracks the progress for the opening of the congestion window by bytes, not segments.

Note that [RFC2861] and [RFC2988] do not address the issue of RTO validation. This is also a problem, particularly when the Congestion Manager [RFC3124] is implemented. During periods of high packet loss, the RTO may be repeatedly increased via exponential back-off, and may attain a high value. Due to lack of timely feedback on RTT and RTO during application-driven periods, the high RTO estimate may persist long after the conditions that generated it have dissipated.

RTO validation MAY be used to address this issue for TCP, via the following procedure:

After the congestion window is decayed according to [RFC2861], reset the estimated RTO to 3 seconds. After the next packet comes in, re-calculate RTTavg, RTTdev, and RTO according to the method described in [RFC2581].

To address this issue for SCTP, AAA implementations SHOULD use SCTP heartbeats. [RFC2960] states that heartbeats should be enabled by default, with an interval of 30 seconds. If this interval proves to be too long to resolve this issue, AAA implementations MAY reduce the heartbeat interval.

### 3.7. Inability to Use Fast Re-Transmit

When Congestion Window Validation [RFC2861] is used, AAA implementations will operate with a congestion window equal to the initial window much of the time. As a result, the window size will often not be large enough to enable use of fast re-transmit for TCP. In addition, since AAA traffic is two-way, ACKs carrying data will not count towards triggering fast re-transmit. SCTP is less likely to encounter this issue, so the measures described below apply to TCP.

To address this issue, AAA implementations SHOULD support selective acknowledgement as described in [RFC2018] and [RFC2883]. AAA implementations SHOULD also implement Limited Transmit for TCP, as described in [RFC3042]. Rather than reducing the number of duplicate ACKs required for triggering fast recovery, which would increase the number of inappropriate re-transmissions, Limited Transmit enables the window size be increased, thus enabling the sending of additional packets which in turn may trigger fast re-transmit without a change to the algorithm.

However, if congestion window validation [RFC2861] is implemented, this proposal will only have an effect in situations where the time between packets is less than the estimated retransmission timeout (RTO). If the time between packets is greater than RTO, additional packets will typically not be available for sending so as to take advantage of the increased window size. As a result, AAA protocols will typically operate with the lowest possible congestion window size, resulting in a re-transmission timeout for every lost packet.

### 3.8. Head of Line Blocking

TCP inherently does not provide a solution to the head-of-line blocking problem, although its effects can be lessened by implementation of Limited Transmit [RFC3042], and connection load balancing.

#### 3.8.1. Using SCTP Streams to Prevent Head of Line Blocking

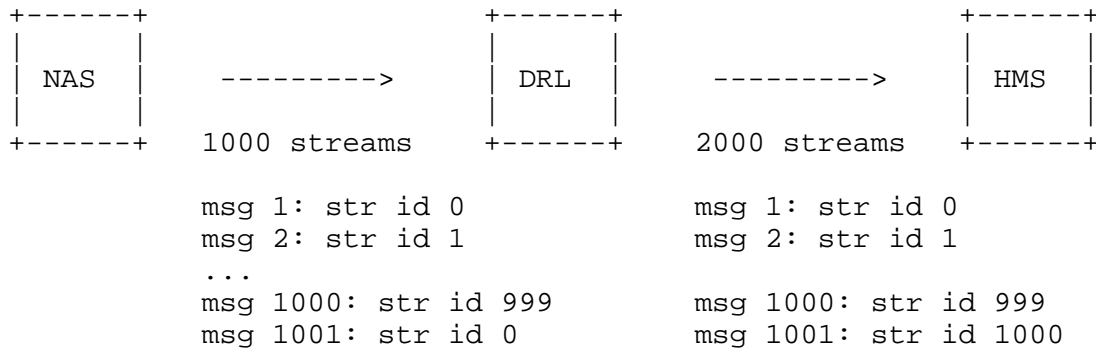
Each AAA node SHOULD distribute its messages evenly across the range of SCTP streams that it and its peer have agreed upon. (A lost message in one stream will not cause any other streams to block.) A trivial and effective implementation of this simply increments a counter for the stream ID to send on. When the counter reaches the maximum number of streams for the association, it resets to 0.

AAA peers MUST be able to accept messages on any stream. Note that streams are used *\*solely\** to prevent head-of-the-line blocking. All identifying information is carried within the Diameter payload. Messages distributed across multiple streams may not be received in the order they are sent.

SCTP peers can allocate up to 65535 streams for an association. The cost for idle streams may or may not be zero, depending on the implementation, and the cost for non-idle streams is always greater than 0. So administrators may wish to limit the number of possible streams on their diameter nodes according to the resources (i.e. memory, CPU power, etc.) of a particular node.

On a Diameter client, the number of streams may be determined by the maximum number of peak users on the NAS. If a stream is available per user, then this should be sufficient to prevent head-of-line blocking. On a Diameter proxy, the number of streams may be determined by the maximum number of peak sessions in progress from that proxy to each downstream AAA server.

Stream IDs do not need to be preserved by relay agents. This simplifies implementation, as agents can easily handle forwarding between two associations with different numbers of streams. For example, consider the following case, where a relay server DRL forwards messages between a NAS and a home server, HMS. The NAS and DRL have agreed upon 1000 streams for their association, and DRL and HMS have agreed upon 2000 streams for their association. The following figure shows the message flow from NAS to HMS via DRL, and the stream ID assignments for each message:



DRL can forward messages 1 through 1000 to HMS using the same stream ID that NAS used to send to DRL. However, since the NAS / DRL association has only 1000 streams, NAS wraps around to stream ID 0 when sending message 1001. The DRL / HMS association, on the other hand, has 2000 streams, so DRL can reassign message 1001 to stream ID 1000 when forwarding it on to HMS.

This distribution scheme acts like a hash table. It is possible, yet unlikely, that two messages will end up in the same stream, and even less likely that there will be message loss resulting in blocking when this happens. If it does turn out to be a problem, local administrators can increase the number of streams on their nodes to improve performance.

### 3.9. Congestion Avoidance

In order to improve upon default timer estimates, AAA implementations MAY implement the Congestion Manager (CM) [RFC3124]. CM is an end-system module that:

- (i) Enables an ensemble of multiple concurrent streams from a sender destined to the same receiver and sharing the same congestion properties to perform proper congestion avoidance and control, and
- (ii) Allows applications to easily adapt to network congestion.

The CM helps integrate congestion management across all applications and transport protocols. The CM maintains congestion parameters (available aggregate and per-stream bandwidth, per-receiver round-trip times, etc.) and exports an API that enables applications to learn about network characteristics, pass information to the CM, share congestion information with each other, and schedule data transmissions.

The CM enables the AAA application to access transport parameters (RTTavg, RTTdev) via callbacks. RTO estimates are currently not available via the callback interface, though they probably should be. Where available, transport parameters SHOULD be used to improve upon default timer values.

### 3.10. Premature Failover

Premature failover is prevented by the watchdog functionality described above. If the next hop does not return a reply, the AAA client will send a watchdog message to it to verify liveness. If a watchdog reply is received, then the AAA client will know that the next hop server is functioning at the application layer. As a result, it is only necessary to provide terminal error messages, such as the following:

"Busy": agent/Server too busy to handle additional requests, NAS should failover all requests to another agent/server.

"Can't Locate": agent can't locate the AAA server for the indicated realm; NAS should failover that request to another proxy.

"Can't Forward": agent has tried both primary and secondary AAA servers with no response; NAS should failover the request to another agent.

Note that these messages differ in their scope. The "Busy" message tells the NAS that the agent/server is too busy for ANY request. The "Can't Locate" and "Can't Forward" messages indicate that the ultimate destination cannot be reached or isn't responding, implying per-request failover.

## 4. Security Considerations

Since AAA clients, agents and servers serve as network access gatekeepers, they are tempting targets for attackers. General security considerations concerning TCP congestion control are discussed in [RFC2581]. However, there are some additional considerations that apply to this specification.

By enabling failover between AAA agents, this specification improves the resilience of AAA applications. However, it may also open avenues for denial of service attacks.

The failover algorithm is driven by lack of response to AAA requests and watchdog packets. On a lightly loaded network where AAA responses would not be received prior to expiration of the watchdog



timer, an attacker can swamp the network, causing watchdog packets to be dropped. This will cause the AAA client to switch to another AAA agent, where the attack can be repeated. By causing the AAA client to cycle between AAA agents, service can be denied to users desiring network access.

Where TLS [RFC2246] is being used to provide AAA security, there will be a vulnerability to spoofed reset packets, as well as other transport layer denial of service attacks (e.g. SYN flooding). Since SCTP offers improved denial of service resilience compared with TCP, where AAA applications run over SCTP, this can be mitigated to some extent.

Where IPsec [RFC2401] is used to provide security, it is important that IPsec policy require IPsec on incoming packets. In order to enable a AAA client to determine what security mechanisms are in use on an agent or server without prior knowledge, it may be tempting to initiate a connection in the clear, and then to have the AAA agent respond with IKE [RFC2409]. While this approach minimizes required client configuration, it increases the vulnerability to denial of service attack, since a connection request can now not only tie up transport resources, but also resources within the IKE implementation.

## 5. IANA Considerations

This document does not create any new number spaces for IANA administration.

## References

### 6.1. Normative References

- [RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC896] Nagle, J., "Congestion Control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC2486] Aboba, B. and M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.
- [RFC2581] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., Podolsky, M. and A. Romanow, "An Extension to the Selective Acknowledgment (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.
- [RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.
- [RFC3042] Allman, M., Balakrishnan H. and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC3074] Volz, B., Gonczi, S., Lemon, T. and R. Stevens, "DHC Load Balancing Algorithm", RFC 3074, February 2001.
- [RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.

## 6.2. Informative References

- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.
- [RFC2401] Atkinson, R. and S. Kent, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.
- [RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.
- [RFC2607] Aboba, B. and J. Vollbrecht, "Proxy Chaining and Policy Implementation in Roaming", RFC 2607, June 1999.
- [RFC2861] Handley, M., Padhye, J. and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC2865] Rigney, C., Willens, S., Rubens, A. and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

- [RFC2866] Rigney, C., "RADIUS Accounting", RFC 2866, June 2000.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC2975] Aboba, B., Arkko, J. and D. Harrington, "Introduction to Accounting Management", RFC 2975, June 2000.
- [RFC3390] Allman, M., Floyd, S. and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.
- [Congest] Jacobson, V., "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988. <ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z>
- [Paxson] Paxson, V., "Measurement and Analysis of End-to-End Internet Dynamics", Ph.D. Thesis, Computer Science Division, University of California, Berkeley, April 1997.

## Appendix A - Detailed Watchdog Algorithm

In this Appendix, the memory control structure that contains all information regarding a specific peer is referred to as a Peer Control Block, or PCB. The PCB contains the following fields:

## Status:

OKAY: The connection is up  
SUSPECT: Failover has been initiated on the connection.  
DOWN: Connection has been closed.  
REOPEN: Attempting to reopen a closed connection  
INITIAL: The initial state of the pcb when it is first created.  
The pcb has never been opened.

## Variables:

Pending: Set to TRUE if there is an outstanding unanswered watchdog request  
Tw: Watchdog timer value  
NumDWA: Number of DWAs received during REOPEN

Tw is the watchdog timer, measured in seconds. Every second, Tw is decremented. When it reaches 0, the OnTimerElapsed event (see below) is invoked. Pseudo-code for the algorithm is included on the following pages.

## SetWatchdog()

```
{
/*
SetWatchdog() is called whenever it is necessary
to reset the watchdog timer Tw. The value of the
watchdog timer is calculated based on the default
initial value TWINIT and a jitter ranging from
-2 to 2 seconds. The default for TWINIT is 30 seconds,
and MUST NOT be set lower than 6 seconds.
*/
    Tw=TWINIT -2.0 + 4.0 * random() ;
    SetTimer(Tw) ;
    return ;
}
```

```
/*
```

```
OnReceive() is called whenever a message
is received from the peer. This message MAY
be a request or an answer, and can include
DWR and DWA messages. Pending is assumed to
be a global variable.
```

```
*/
```

```
OnReceive(pcb, msgType)
```

```

{
    if (msgType == DWA) {
        Pending = FALSE;
    }
    switch (pcb->Status){
    case OKAY:
        SetWatchdog();
        break;
    case SUSPECT:
        pcb->Status = OKAY;
        Failback(pcb);
        SetWatchdog();
        break;
    case REOPEN:
        if (msgType == DWA) {
            NumDWA++;
            if (NumDWA == 3) {
                pcb->status = OKAY;
                Failback();
            }
        } else {
            Throwaway(received packet);
        }
        break;
    case INITIAL:
    case DOWN:
        Throwaway(received packet);
        break;
    default:
        Error("Shouldn't be here!");
        break;
    }
}

/*
OnTimerElapsed() is called whenever Tw reaches zero (0).
*/
OnTimerElapsed(pcb)
{
    switch (pcb->status){
    case OKAY:
        if (!Pending) {
            SendWatchdog(pcb);
            SetWatchdog();
            Pending = TRUE;
            break;
        }
        pcb->status = SUSPECT;
    }
}

```

```

        FailOver(pcb);
        SetWatchdog();
        break ;
    case SUSPECT:
        pcb->status = DOWN;
        CloseConnection(pcb);
        SetWatchdog();
        break;
    case INITIAL:
    case DOWN:
        AttemptOpen(pcb);
        SetWatchdog();
        break;
    case REOPEN:
        if (!Pending) {
            SendWatchdog(pbc);
            SetWatchdog();
            Pending = TRUE;
            break;
        }
        if (NumDWA < 0) {
            pcb->status = DOWN;
            CloseConnection(pcb);
        } else {
            NumDWA = -1;
        }
        SetWatchdog();
        break;
    default:
        error("Shouldn't be here!");
        break;
    }
}

/*
OnConnectionUp() is called whenever a connection comes up
*/
OnConnectionUp(pcb)
{
    switch (pcb->status){
        case INITIAL:
            pcb->status = OKAY;
            SetWatchdog();
            break;
        case DOWN:
            pcb->status = REOPEN;
            NumDWA = 0;
            SendWatchdog(pcb);

```

```

        SetWatchdog();
        Pending = TRUE;
        break;
    default:
        error("Shouldn't be here!");
        break;
    }
}

/*
OnConnectionDown() is called whenever a connection goes down
*/
OnConnectionDown(pcb)
{
    pcb->status = DOWN;
    CloseConnection();
    switch (pcb->status){
        case OKAY:
            Failover(pcb);
            SetWatchdog();
            break;
        case SUSPECT:
        case REOPEN:
            SetWatchdog();
            break;
        default:
            error("Shouldn't be here!");
            break;
    }
}

```

/\* Here is the state machine equivalent to the above code:

STATE =====	Event -----	Actions -----	New State -----
OKAY	Receive DWA	Pending = FALSE SetWatchdog()	OKAY
OKAY	Receive non-DWA	SetWatchdog()	OKAY
SUSPECT	Receive DWA	Pending = FALSE Failback() SetWatchdog()	OKAY
SUSPECT	Receive non-DWA	Failback() SetWatchdog()	OKAY
REOPEN	Receive DWA & NumDWA == 2	Pending = FALSE NumDWA++ Failback()	OKAY
REOPEN	Receive DWA & NumDWA < 2	Pending = FALSE NumDWA++	REOPEN

STATE	Event	Actions	New State
=====	-----	-----	-----
REOPEN	Receive non-DWA	Throwaway()	REOPEN
INITIAL	Receive DWA	Pending = FALSE Throwaway()	INITIAL
INITIAL	Receive non-DWA	Throwaway()	INITIAL
DOWN	Receive DWA	Pending = FALSE Throwaway()	DOWN
DOWN	Receive non-DWA	Throwaway()	DOWN
OKAY	Timer expires & !Pending	SendWatchdog() SetWatchdog() Pending = TRUE	OKAY
OKAY	Timer expires & Pending	Failover() SetWatchdog()	SUSPECT
SUSPECT	Timer expires	CloseConnection() SetWatchdog()	DOWN
INITIAL	Timer expires	AttemptOpen() SetWatchdog()	INITIAL
DOWN	Timer expires	AttemptOpen() SetWatchdog()	DOWN
REOPEN	Timer expires & !Pending	SendWatchdog() SetWatchdog() Pending = TRUE	REOPEN
REOPEN	Timer expires & Pending & NumDWA < 0	CloseConnection() SetWatchdog()	DOWN
REOPEN	Timer expires & Pending & NumDWA >= 0	NumDWA = -1 SetWatchdog()	REOPEN
INITIAL	Connection up	SetWatchdog() NumDWA = 0 SendWatchdog() SetWatchdog()	OKAY
DOWN	Connection up	Pending = TRUE CloseConnection() Failover() SetWatchdog()	REOPEN
OKAY	Connection down	CloseConnection() Failover() SetWatchdog()	DOWN
SUSPECT	Connection down	CloseConnection() SetWatchdog()	DOWN
REOPEN	Connection down	CloseConnection() SetWatchdog()	DOWN

\*/



## Appendix B - AAA Agents

As described in [RFC2865] and [RFC2607], AAA agents have become popular in order to support services such as roaming and shared use networks. Such agents are used both for authentication/authorization, as well as accounting [RFC2975].

AAA agents include:

- Relays
- Proxies
- Re-directs
- Store and Forward proxies
- Transport layer proxies

The transport layer behavior of each of these agents is described below.

### B.1 Relays and Proxies

While the application-layer behavior of relays and proxies are different, at the transport layer the behavior is similar. In both cases, two connections are established: one from the AAA client (NAS) to the relay/proxy, and another from the relay/proxy to the AAA server. The relay/proxy does not respond to a client request until it receives a response from the server. Since the two connections are de-coupled, the end-to-end conversation between the client and server may not self clock.

Since AAA transport is typically application-driven, there is frequently not enough traffic to enable ACK piggybacking. As a result, the Nagle algorithm is rarely triggered, and delayed ACKs may comprise nearly half the traffic. Thus AAA protocols running over reliable transport will see packet traffic nearly double that experienced with UDP transport. Since ACK parameters (such as the value of the delayed ACK timer) are typically fixed by the TCP implementation and are not tunable by the application, there is little that can be done about this.

A typical trace of a conversation between a NAS, proxy and server is shown below:

Time	NAS	Relay/Proxy	Server
-----	---	-----	-----
0	Request ----->		
OTTnp + Tpr		Request ----->	
OTTnp + TdA		Delayed ACK <-----	
OTTnp + OTTps + Tpr + Tsr			Reply/ACK <-----
OTTnp + OTTps + Tpr + Tsr + OTTsp + TpR		Reply <-----	
OTTnp + OTTps + Tpr + Tsr + OTTsp + TdA		Delayed ACK ----->	
OTTnp + OTTps + OTTsp + OTTpn + Tpr + Tsr + TpR + TdA	Delayed ACK ----->		

#### Key

---

OTT = One-way Trip Time  
 OTTnp = One-way trip time (NAS to Relay/Proxy)  
 OTTpn = One-way trip time (Relay/Proxy to NAS)  
 OTTps = One-way trip time (Relay/Proxy to Server)  
 OTTsp = One-way trip time (Server to Relay/Proxy)  
 TdA = Delayed ACK timer  
 Tpr = Relay/Proxy request processing time  
 TpR = Relay/Proxy reply processing time  
 Tsr = Server request processing time

At time 0, the NAS sends a request to the relay/proxy. Ignoring the serialization time, the request arrives at the relay/proxy at time OTTnp, and the relay/proxy takes an additional Tpr in order to forward the request toward the home server. At time TdA after

receiving the request, the relay/proxy sends a delayed ACK. The delayed ACK is sent, rather than being piggybacked on the reply, as long as  $TdA < OTTps + OTTsp + Tpr + Tsr + TpR$ .

Typically  $Tpr < TdA$ , so that the delayed ACK is sent after the relay/proxy forwards the request toward the server, but before the relay/proxy receives the reply from the server. However, depending on the TCP implementation on the relay/proxy and when the request is received, it is also possible for the delayed ACK to be sent prior to forwarding the request.

At time  $OTTnp + OTTps + Tpr$ , the server receives the request, and  $Tsr$  later, it generates the reply. Where  $Tsr < TdA$ , the reply will contain a piggybacked ACK. However, depending on the server responsiveness and TCP implementation, the ACK and reply may be sent separately. This can occur, for example, where a slow database or storage system must be accessed prior to sending the reply.

At time  $OTTnp + OTTps + OTTsp + Tpr + Tsr$  the reply/ACK reaches the relay/proxy, which then takes  $TpR$  additional time to forward the reply to the NAS. At  $TdA$  after receiving the reply, the relay/proxy generates a delayed ACK. Typically  $TpR < TdA$  so that the delayed ACK is sent to the server after the relay/proxy forwards the reply to the NAS. However, depending on the circumstances and the relay/proxy TCP implementation, the delayed ACK may be sent first.

As with a delayed ACK sent in response to a request, which may be piggybacked if the reply can be received quickly enough, piggybacking of the ACK sent in response to a reply from the server is only possible if additional request traffic is available. However, due to the high inter-packet spacings in typical AAA scenarios, this is unlikely unless the AAA protocol supports a reply ACK.

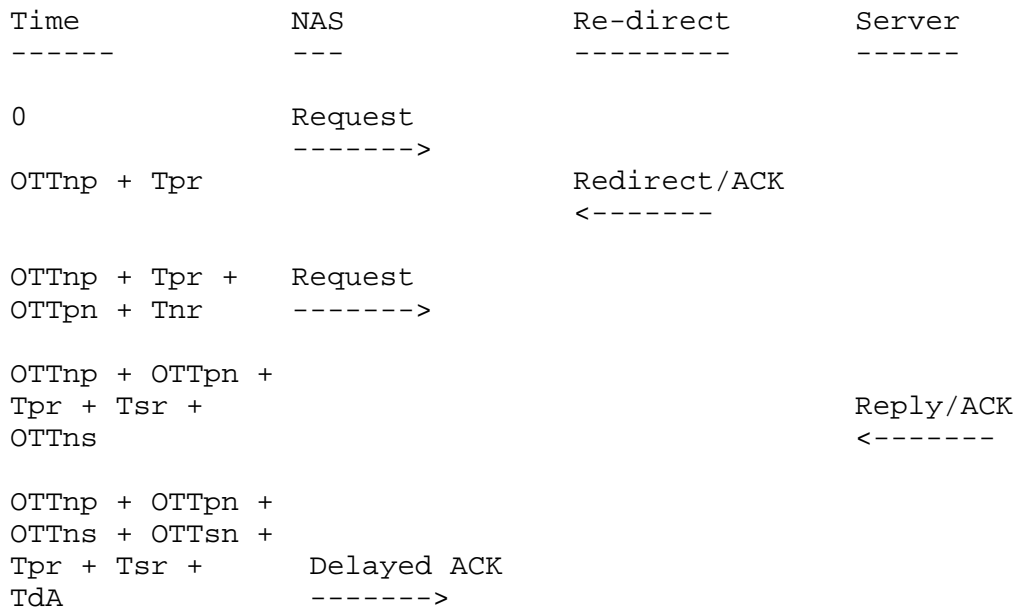
At time  $OTTnp + OTTps + OTTsp + OTTpn + Tpr + Tsr + TpR$  the NAS receives the reply.  $TdA$  later, a delayed ACK is generated.

## B.2 Re-directs

Re-directs operate by referring a NAS to the AAA server, enabling the NAS to talk to the AAA server directly. Since a direct transport connection is established, the end-to-end connection will self-clock.

With re-directs, delayed ACKs are less frequent than with application-layer proxies since the Re-direct and Server will typically piggyback replies with ACKs.

The sequence of events is as follows:



Key

---

OTT = One-way Trip Time  
 OTTnp = One-way trip time (NAS to Re-direct)  
 OTTpn = One-way trip time (Re-direct to NAS)  
 OTTns = One-way trip time (NAS to Server)  
 OTTsn = One-way trip time (Server to NAS)  
 TdA = Delayed ACK timer  
 Tpr = Re-direct processing time  
 Tnr = NAS re-direct processing time  
 Tsr = Server request processing time

### B.3 Store and Forward Proxies

With a store and forward proxy, the proxy may send a reply to the NAS prior to forwarding the request to the server. While store and forward proxies are most frequently deployed for accounting [RFC2975], they also can be used to implement authentication/authorization policy, as described in [RFC2607].

As noted in [RFC2975], store and forward proxies can have a negative effect on accounting reliability. By sending a reply to the NAS without receiving one from the accounting server, store and forward proxies fool the NAS into thinking that the accounting request had been accepted by the accounting server when this is not the case. As a result, the NAS can delete the accounting packet from non-volatile

storage before it has been accepted by the accounting server. That leaves the proxy responsible for delivering accounting packets. If the proxy involves moving parts (e.g. a disk drive) while the NAS does not, overall system reliability can be reduced. As a result, store and forward proxies SHOULD NOT be used.

The sequence of events is as follows:

Time -----	NAS ---	Proxy -----	Server -----
0	Request ----->		
OTTnp + TpR		Reply/ACK -----<	
OTTnp + Tpr		Request ----->	
OTTnp + OTTph + Tpr + Tsr			Reply/ACK -----<
OTTnp + OTTph + Tpr + Tsr + OTThp + TpR		Reply -----<	
OTTnp + OTTph + Tpr + Tsr + OTThp + TdA		Delayed ACK ----->	
OTTnp + OTTph + OTThp + OTTpn + Tpr + Tsr + TpR + TdA	Delayed ACK ----->		

Key

---

OTT = One-way Trip Time  
 OTTnp = One-way trip time (NAS to Proxy)  
 OTTpn = One-way trip time (Proxy to NAS)  
 OTTph = One-way trip time (Proxy to Home server)  
 OTThp = One-way trip time (Home Server to Proxy)  
 TdA = Delayed ACK timer  
 Tpr = Proxy request processing time  
 TpR = Proxy reply processing time  
 Tsr = Server request processing time

## B.4 Transport Layer Proxies

In addition to acting as proxies at the application layer, transport layer proxies forward transport ACKs between the AAA client and server. This splices together the client-proxy and proxy-server connections into a single connection that behaves as though it operates end-to-end, exhibiting self-clocking. However, since transport proxies operate at the transport layer, they cannot be implemented purely as applications and they are rarely deployed.

With a transport proxy, the sequence of events is as follows:

Time -----	NAS ---	Proxy -----	Home Server -----
0	Request ----->		
OTTnp + Tpr		Request ----->	
OTTnp + OTTph + Tpr + Tsr			Reply/ACK <-----
OTTnp + OTTph + Tpr + Tsr + OTThp + TpR		Reply/ACK <-----	
OTTnp + OTTph + OTThp + OTTpn + Tpr + Tsr + TpR + TdA	Delayed ACK ----->		
OTTnp + OTTph + OTThp + OTTpn + Tpr + Tsr + TpR + TpD		Delayed ACK ----->	

## Key

---

OTT = One-way Trip Time  
 OTTnp = One-way trip time (NAS to Proxy)  
 OTTpn = One-way trip time (Proxy to NAS)  
 OTTph = One-way trip time (Proxy to Home server)  
 OTThp = One-way trip time (Home Server to Proxy)  
 TdA = Delayed ACK timer  
 Tpr = Proxy request processing time  
 TpR = Proxy reply processing time

Tsr = Server request processing time  
TpD = Proxy delayed ack processing time

#### Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

#### Acknowledgments

Thanks to Allison Mankin of AT&T, Barney Wolff of Databus, Steve Rich of Cisco, Randy Bush of AT&T, Bo Landarv of IP Unplugged, Jari Arkko of Ericsson, and Pat Calhoun of Blackstorm Networks for fruitful discussions relating to AAA transport.

## Authors' Addresses

Bernard Aboba  
Microsoft Corporation  
One Microsoft Way  
Redmond, WA 98052

Phone: +1 425 706 6605  
Fax: +1 425 936 7329  
EMail: bernarda@microsoft.com

Jonathan Wood  
Sun Microsystems, Inc.  
901 San Antonio Road  
Palo Alto, CA 94303

EMail: jonwood@speakeasy.net



## Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

## Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

