NOTICE

© 2005 RADVISION Ltd. All intellectual property rights in this publication are owned by RADVISION Ltd. and are protected by United States copyright laws, other applicable copyright laws and international treaty provisions. RADVISION Ltd. retains all rights not expressly granted.

The publication is RADVISION confidential. No part of this publication may be reproduced in any form whatsoever or used to make any derivative work without prior written approval by RADVISION Ltd. No representation of warranties for fitness for any purpose other than what is specifically mentioned in this guide is made either by RADVISION Ltd. or its agents.

RADVISION Ltd. reserves the right to revise this publication and make changes without obligation to notify any person of such revisions or changes. RADVISION Ltd. may make improvements or changes in the product(s) and/or the program(s) described in this documentation at any time. If there is any software on removable media described in this publication, it is furnished under a license agreement included with the product as a separate document. If you are unable to locate a copy, please contact RADVISION Ltd. and a copy will be provided to you.

Unless otherwise indicated, RADVISION registered trademarks are registered in the United States and other territories. All registered trademarks recognized.

For further information contact RADVISION or your local distributor or reseller.

H.323 Stack Test Application version 5.0, June, 2005
Publication 2
http://www.radvision.com
CONTENTS

Troubleshooting 1

Development Process 1
  Finding Errors and Warnings 1
  Finding Resource Leaks 2
  Writing Automatic Resource Checking in the Application 4
  Caution 6
  rvtele.ini and ms API Usage 7

Troubleshooting 9
  1. General questions 9
  2. Resource Leaks 11
  3. General Indications in the Log 18
  4. Stack Crashes 19
  5. Deadlock in a Multi-threaded Application 22
  6. Message Received by Other Side Is Different Than What Appears in TPKTCHAN or UDPCHAN 23
  7. Received Message Different Than What Appears To Sniffer 24
  8. Information Needed by Customer Support 26
  Older H.323 Stack versions 30

Protecting An Application Against DoS Attacks 32
  Aliases 33
  Object Identifiers 35
  Strings 36
TROUBLESHOOTING

The RADVISION H.323 Protocol Stack is a complex toolkit with a rich set of features. When writing applications on top of such a stack, troubleshooting and debugging are key issues.

This guide helps you learn the best and easiest ways to debug and troubleshoot your code, allowing you to find resource leaks in a short period of time without the need for long scenarios. It also gives you tools to help you communicate easily with RADVISION's Customer Support team, so as to shorten the time it takes RADVISION to solve open issues and problems.

This guide includes the following sections:

- Development Process
- Troubleshooting
- Protecting An Application Against DoS Attacks

DEVELOPMENT PROCESS

Troubleshooting and debugging begins in your development process. To find problems at this initial stage, you should make use of two important Stack features: Log and Watchdog.

FINDING ERRORS AND WARNINGS

Whenever you are running in debug mode—during the development process, in QA testing, or in operational environments where a log is applicable—it is highly recommended that errors and exceptions be displayed in the Log at all times.

To effect this, add the following line to the rvtele.ini file as the first line in the list of sources that you are filtering to the Log:

* = ERROR EXCEPTION

While you are developing software on top of the Stack, you may also want to add filter warnings to get more detailed information about Stack and application behavior. In this case, use:

* = ERROR EXCEPTION WARNING

1. An example of a clean rvtele.ini file can be found at the end of this chapter.
Development Process

Check the resulting log file to make sure you know what the errors, exceptions and warnings in the Log indicate—they could indicate a problem in your code or in the Stack build you are using.

If you do not use the rvtele.ini file, but you still use logging (for example, some of RADVISION’s embedded platform customers do not have a file system available), you can call:

```c
msAdd("*=ERROR EXCEPTION WARNING");
```

The results will be the same as placing this line in the rvtele.ini file.  

FINDING RESOURCE LEAKS

Now that you are checking for errors and warnings on a regular basis, you can also add another stabilizing factor to your application by making sure you check for possible resource leaks. You can do this during your development and QA testing phases, or you can write a module inside your application to automatically check for such problems during runtime.

Finding resource leaks in the Stack has been made easy by the introduction of the Watchdog module. This module is responsible for keeping track of most allocated resources of the Stack and helps you find resource leaks. This includes finding PVT nodes that were never deleted, calls and RAS transactions that were never closed, etc. The Watchdog itself uses a log source called “WATCHDOG”.

To produce Watchdog output in the Log, you can add the following line to the rvtele.ini file:

```
"WATCHDOG=1"
```

or use: `msAdd("WATCHDOG");`

**Note** An example of the above can be found at the end of this chapter.

The Watchdog displays information on three different occasions:

- **On cmInitialize() and in cmEnd().** This helps you see any resource leaks that might have happened during the whole time the application was active.
- **On a given interval.** This shows you if there have been any changes in resources over time.

---

1. An example of ms function calls for setting log parameters can be found in the section `rvtele.ini` and `ms API Usage`.

---

2 H.323 Protocol Stack Troubleshooting Guide
Upon application request. The Watchdog API allows you to print the information it contains to the Log or pass it to your application for any other purpose.

**Note**  For detailed information about the Watchdog API and output format, please see the *RADVISION H.323 Protocol Stack Programmer Guide*, Watchdog section.

For each resource, the information displayed by the Watchdog includes:

- The value with which it started
- The current number, if elements from the resource are being used
- The peak number of elements from the resource that have been used so far
- The maximum number of elements it can allocate.

This information enables you to find resources that were not released and that might cause a resource problem later on in long testing scenarios. You can also use this information to fine-tune the resource allocations you need, so as to reduce the memory requirements of your application.

If you suspect a resource leak or want to make sure that none exists, you can configure the Stack to print resource information to the Log at a given interval. After running your scenario, wait a few minutes for the Stack to cool down (some of the resources will only be de-allocated after a period of time) and check the printout at the interval. Compare the values in that printout to the ones from just before the scenario started to see if you can locate a problem. For a sample scenario debugging session, see the Example below.

The most common indication of a resource leak is that a long scenario will display an "RA full" message. An example of such an error is:

```
H323Main 19:51:30 RA       : ERROR - raAdd (VT tree): Array full (6150 elements)
```

The "VT tree" indication displays the name of the resource that is probably leaking, with the total number of elements allocated for it. In most cases, the problem can be found in shorter scenarios by using the Watchdog.

You can also add some resource-checking features to your application. For example, this can serve to send alerts from your applications when they run at customer sites. The GUI-based sample test application that is supplied with the Stack does some "automatic" resource checking of its own. The resources

**Writing Automatic Resource Checking in the Application**
Development Process

window that can be opened from the toolbar is implemented on top of the Watchdog API (for more information, see Status_Display function in TAP_tools.c source code of the sample test application).

The test application also has an automatic check that is invoked if the test application is left open for a full minute with no open calls in it. For more information, see Status_Check function in TAP_tools.c source code of the sample test application. This shows how the check is made on specific resources that interest you.

EXAMPLE

The Log printout below shows an example of a scenario with a resource leak and some errors.

This scenario was produced by doing the following:

1. Removing the call cmCallClose(), which releases a "CmCalls" resource from the code of the sample test application. This function was called from cmEvCallStateChanged on the state cmCallStateIdle.

2. Calling to an IP address that is not listening for incoming calls.

Note  Some of the messages that should appear in the Log have been removed to shorten the Log. The Log formatting has been modified to fit the size of the page.

The Log was created by using the following rvtele.ini file:

```
[supserve]
debLevel=1
msgfile=2

[insertIntoFile]
*=ERROR WARNING EXCEPT
TPKITCHAN=1
WATCHDOG=1
CMAPI=1
CMAPICB=1
```
In reviewing this log file, please note the following:

- Lines 2-15: Before cmInitialize() returns, it prints out the Watchdog resource information. The initial and current value of cmValTree is 254.

  The maximum number of calls that can be dialed is 20 (line 12).

- Lines 25, 26, 31, 32: Since errors are caught from all log sources, Transport and SOCKET error messages are displayed in the Log. This is because you are dialing an address that is not listening for incoming calls.
Lines 28-29: You reach cmCallStateIdle state on the outgoing call, and nowhere in the Log is there a call to cmCallClose(). This is the cause of the resource leak we are trying to locate.

Line 35: cmEnd() was called and the Watchdog prints out its information.

Lines 37, 45: TransportSessions and CmCalls are left with an open resource: You have a possible leak.

Line 46: CmValTree has 536 PVT nodes rather than the initial 254. This may also be a resource leak (and it is, since the additional nodes hold information about the call that was never closed).

Lines 49-51: There are warnings when the Stack is closed, because some elements were never deleted. If you are looking at such a log—after finding error messages and checking the latest Watchdog printout—you should notice that there are resource leaks as well. At this stage, first try and find out if the resources are really leaking and resolve the issue. Later you should see if the errors are the cause of the problems and fix them, or ignore them if they fit the scenario you are running.

The same information can be seen and checked using the timeout interval or the explicit log-printing API function of the Watchdog (RvH323WatchdogPrint).

**CAUTION**

When using the Watchdog to check resources, remember the following:

- Registration to a gatekeeper uses up resources. If you are registered, you should expect some resources to be used up on top of the initial values (this includes IncomingRasTx, OutgoingRasTx, RasTimers, CmTimers and CmValTree). If you want your resource checking test during debugging to be more accurate, you may want to unregister from the gatekeeper as well (using cmUnregister) and wait a few minutes before starting to check resources.

- Incoming RAS transactions are not closed when you call cmRASCslose(), but are closed using a garbage collection mechanism. This mechanism is called whenever new incoming RAS transactions are found and when the Stack is stopped. This means that effectively, the incoming RAS transaction resources will be higher than 0 if any incoming transaction was received. The best way of validating that there are no resource leaks in
Development Process

incoming RAS transactions, is waiting several seconds and then sending a RAS transaction to the checked application and validating that there is only a single incoming RAS transaction currently open.

- Multiplexed TCP connections that are maintained when no calls are opened on them, act somewhat similar to incoming RAS transactions. This means that these connections will be automatically garbage-collected when you call cmStop() or cmEnd(). Multiplexed TCP connections usually take up TransportHosts resources.

- If you are using the H.450 Supplementary Services Add-on, it should be linked to the Watchdog by using sseWatchdogInit() to make sure it also checks and displays the resources of this module.

RVTELE.INI AND MS API USAGE

The following is an example of an rvtele.ini file that is suggested for use in most debugging and development scenarios:

```
[supserve]
debLevel=2
msgfile=2
flushfile=0
#filesize=20000000
#numoffiles=10
#filename=mylogfile.log

[insertIntoFile]
*=ERROR WARNING EXCEPTION
TPKTCHAN=1
UDPCHAN=1
FASTSTART=1
WATCHDOG=1
CONFIG=1
CMAPI=1
CMAPICB=1
UNREG=1
```
The commented lines above allow the use of cyclical log files of a specific size with a name different than the default rvtsp.log.

To get these same results using the ms API functions:

```c
msSetDebugLevel(2);
msSetStackNotify(myLogFunction);
msOpen();

msAdd("*=ERROR WARNING EXCEPTION");
msAdd("TPKTCHAN");
msAdd("UDPCHAN");
msAdd("FASTSTART");
msAdd("WATCHDOG");
msAdd("CONFIG");
msAdd("CMAPI");
msAdd("CMAPICB");
msAdd("UNREG");
```

The main difference between the two examples above is that the ms functions are logged to a given function implemented by the application, while rvtele.ini sends the Log messages to a file.
This section describes the information the RADVISION Customer Support team needs from you if you are experiencing a problem. The following list of questions is a good way to begin working on a problem and may greatly reduce the time it takes RADVISION to solve it.

### 1. General Questions

Go through the list below and see if you can find the problem you are experiencing (or one similar to it).

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 I think there is a resource leak somewhere. Possible symptoms:</td>
<td>Go to 2. Resource Leaks</td>
</tr>
<tr>
<td>After some time, no more calls or RAS transactions can be sent or</td>
<td></td>
</tr>
<tr>
<td>received by the Stack; &quot;Array Full&quot; messages appear in the log file.</td>
<td></td>
</tr>
<tr>
<td>1.2 I got an error/warning/exception in the Log. How do I know if it</td>
<td>Go to 3. General Indications in the Log</td>
</tr>
<tr>
<td>indicates a real problem?</td>
<td></td>
</tr>
<tr>
<td>1.3 My application crashes somewhere in the Stack code.</td>
<td>Go to 4. Stack Crashes</td>
</tr>
<tr>
<td>1.4 I am running a multithreaded application and I have a thread that</td>
<td>Go to 5. Deadlock in a Multi-threaded</td>
</tr>
<tr>
<td>is deadlocked.</td>
<td>Application</td>
</tr>
<tr>
<td>1.5 I sent out a message, but it looks different in the remote endpoint</td>
<td>Go to 6. Message Received by Other Side Is Different Than What Appears in TPKTCHAN or UDPCHAN</td>
</tr>
<tr>
<td>and network sniffer than it does in TPKTCHAN/UDPCHAN log source in the</td>
<td></td>
</tr>
<tr>
<td>sender.</td>
<td></td>
</tr>
</tbody>
</table>
Troubleshooting

1.6 I received a message, but it looks different in the sending endpoint and network sniffer than it does in TPKTCAN/UDPCHAN log source in the receiver.

1.7 The Stack does not send messages, and a PERERR/Q931ERR error message appears in the Log.

Go to 7. Received Message Different Than What Appears To Sniffer

Go to 3. General Indications in the Log
2. RESOURCE LEAKS
You were probably referred to this section because you found an “RA full” message in the Log or because the Watchdog indicated that certain resources had not been released.

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 An &quot;RA full&quot; message is displayed in the standard output or in the Log.</td>
<td>Go to 2.3.</td>
</tr>
<tr>
<td>2.2 I think there is a resource leak. I checked the Watchdog printouts and it seems there are resources that were never freed.</td>
<td>Go to 2.2.1.</td>
</tr>
<tr>
<td>2.2.1 Did you wait for several minutes without any activity before checking the resources?</td>
<td>Yes—Go to 2.2.2. No—Wait for several minutes to see if the resources are freed.</td>
</tr>
<tr>
<td>2.2.2 Are the reported resource errors related to the TransportHosts?</td>
<td>Yes—Go to 2.2.2.1. No—Go to 2.2.3.</td>
</tr>
<tr>
<td>2.2.2.1 I am running with Q.931 multiplexing and I do not close connections without calls (this means I set cmParamShutdownEmptyConnection to RV_FALSE).</td>
<td>Yes—Try calling cmStop(). If the TransportHosts resources are freed, then this is not a resource leak, but TCP connections that are left open for faster call establishment. No—Go to 1.4.</td>
</tr>
<tr>
<td>2.2.3 The resource causing me problems is IncomingRasTx.</td>
<td>Yes—Go to 2.2.3.1. No—Go to 2.2.4.</td>
</tr>
</tbody>
</table>
2.2.3.1 Open CMAPI and CMAPICB in the Log. Make sure that the total number of times that cmEvRASRequest, cmEvAutoRASRequest and cmRASStartTransaction appear in the Log is equal to the number of times cmRASClose appears in the Log.

Same number? Go to I.2.
Different number? Go to 2.2.3.2.

2.2.3.2 Check to see that every transaction indicated by cmEvRASRequest has a cmRASClose for it.

Missing cmRASClose? Add it to your code after confirming or rejecting this transaction. Same number? Go to I.2.

2.2.4 The resource indicates a problem with CmValTree or SsValTree.

Yes—Go to I.3.
No—Go to 2.2.5.

2.2.5 The resource causing me problems is CmTimers.

Yes—Go to 2.2.5.1.
No—Go to 2.2.6.

2.2.5.1 One or two timers are always being used, even though I am not doing anything and no calls are open.

Yes—These resources need timers to run properly. This is not a leak. No—Go to I.5.

2.2.5.2 I am registered to a gatekeeper/I am using the Watchdog interval.

Yes—Go to 2.2.6.1.
No—Go to 2.2.6.

2.2.6 The resource causing me problems is RasTimers or OutgoingRasTx.

Yes—Go to 2.2.6.1.
No—Go to I.1.

2.2.6.1 Are you generating RAS transactions using cmRASStartTransaction manually?

Yes—Go to 2.2.6.2.
No—This means you are using automatic RAS only. Go to I.2.
2.2.6.2 Open CMAPI and CMAPICB in the Log. Make sure that the total number of times that cmEvRASRequest, cmEvAutoRASRequest and cmRASStartTransaction appear in the Log is equal to the number of times cmRASClose appears in the Log.

Same number? Go to I.2.
Different number? Go to 2.2.6.3.

2.2.6.3 Check to see that every transaction initiated by cmRASStartTransaction has a cmRASClose.

Missing cmRASClose? Go to 2.2.6.4.
Same number? Go to I.2.

2.2.6.4 Is the missing cmRASClose to a cmRASStartTransaction you called directly from the application?

Yes—Add the call to cmRASClose somewhere, preferably in the reply callback or the timeout callback (cmEvRASRequest, cmEvRASReject, cmEvRASConfirm, cmEvRASTimeout).
No—Go to I.2.

2.3 I am running a scenario where it takes a long time before the problem shows (30 minutes to several days).

Yes—Go to 2.3.1.
No—Go to 2.4.
Troubleshooting

2.3.1 Try running the same scenario for several minutes or several hundreds of calls, if possible. It also might be helpful to configure fewer resources available for the Stack; for example, if you typically configure the Stack to support 1000 calls, change the number to 100. This should make the problem appear more quickly. Once all calls have been completed and closed, wait several minutes for the system to stabilize and then check the resources using the Watchdog (through the Watchdog interval or the Watchdog API functions).

2.4 It seems that this is a resource problem that might have been found using a Watchdog. The following questions address the type of resource indicated in the "RA full" message.

2.4.1 The "RA full" is for "VT tree".

   Found a leak? Go to 2.2 with the shorter scenario.
   No luck? Try running longer scenarios. The problem should appear eventually—much sooner than it would in the long scenario you are running. When you have the shorter scenario (or the long one if you were not able to shorten it according to the guidelines described above), Go to 2.2.

   Yes—Go to I.3.
   No—Go to 2.4.2.
2.4.2  The "RA full" is for "ApplicationTimers".

Yes—It is possible that you are using mtimerSet() without calling mtimerReset() on timers your application is using. You can also go to 1.7 and Customer Support will help you find the bug in your application.

No—Go to 2.4.3.

2.4.3  The "RA full" is for "SELI USER FDs".

Yes—You are using seliCallOn() on more file descriptors than the Stack allocates by default. You can increase the value allocated for users in h323/middle/seli.c. It is defined using the macro SELI_MAX_USER_FDS.

No—Go to 2.4.4.

2.4.4  The "RA full" is for "CM Calls".

2.4.4.1 Make sure you have no connected calls, by checking the Log (with CMAPI and CMAPICB) and validating that each cmCallNew and cmEvNewCall have a cmEvCallStateChanged with the state cmCallStateIdle.

Yes—Go to 2.4.4.1.

No—Go to 2.4.5.

If you found that some calls did not get the Idle state—Go to 2.4.4.2.

If calls have become idle—Go to 2.4.4.3.
2.4.4.2 Make sure all calls are disconnected in the scenario that is being tested.

Yes, they should all be disconnected—Go to I.4.
No, my mistake—I will probably need to drop some calls before checking the problem again.

2.4.4.3 Make sure you called cmCallClose for all of the calls.

Yes, all calls have cmCallClose—Go to I.4. In this case, the EMA source in the Log can be very useful.
No, I forgot to call cmCallClose on some of them—Call cmCallClose on all idle calls before checking the problem again.

2.4.5 The "RA full" is for "CAT Hash" or "CAT CALLS".

Yes—Go to 2.4.5.1.
No—Go to 2.4.6.

2.4.5.1 Make sure you do not have a "CM Calls" problem in addition (using the Watchdog), or go on and check 2.4.4.1 before continuing here.

2.4.6 The "RA full" is for "RAS OUT TX".

Yes—Go to 2.2.6.1.
No—Go to 2.4.7.

2.4.7 The "RA full" is for "RAS IN TX".

Yes—Go to 2.2.3.1.
No—Go to 2.4.8.

2.4.8 The "RA full" is for "RAS OUT HASH" or "RAS IN HASH".

Yes—Go to I.2.
No—Go to 2.4.9.

2.4.9 The "RA full" is for "RAS MESSAGES".

Yes—Go to I.2.
No—Go to 2.4.10.

2.4.10 The "RA full" is for "Timer Array".

Yes—Go to I.5.
No—Go to 2.4.11.
2.4.11 The "RA full" is for "TRANSPORT Sessions", "TRANSPORT Hosts", "Transport Hosts hash", "Transport Sessions hash", "outgoing/tunneled messages" or "TPKT elements".

Yes—Go to I.4.
No—Go to 2.4.12.

2.4.12 The "RA full" is for a different type of resource (not specified above).

Go to I.1.
### 3. General Indications in the Log

Warnings, errors and exceptions in the Log do not always indicate trouble, but they do point strongly in that direction. If the Stack does not work as expected, the first thing to do is check the Log for any such messages.

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 I have a PERERR/Q931ERR message in the Log.</td>
<td>Yes—Go to 3.1.1.</td>
</tr>
<tr>
<td>3.1.1 I may not have built the message quite right or the received message was syntactically incorrect. Opening TPKTCHAN and UDPCHAN subsystems will help determine if the message was incoming or outgoing.</td>
<td>No—Go to 3.2. Check your code or go to I.1.</td>
</tr>
<tr>
<td>3.2 I have a VT log source warning message in the Log.</td>
<td>Yes—Go to 3.2.1.</td>
</tr>
<tr>
<td>3.2.1 This typically happens when a PVT function is called to build nodes that do not exist in the ASN.1 definition of the message itself.</td>
<td>No—Go to I.1. Check &lt;link to the last chapter regarding older versions&gt; if you are using an older version. If you need help with troubleshooting this problem, go to I.1.</td>
</tr>
</tbody>
</table>
4. **STACK CRASHES**

The Stack can crash for three main reasons: Access of deleted objects, access to NULL pointers, or access to trash addresses. This section helps you find the reason for the crash. Once you have, you may then fix it (if possible) or send an accurate crash report to RADVISION (be sure to send both a log and a call stack of the same crash, and if possible, the line of the crash and the values of parameters in that line).

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Can the crash be reproduced?</td>
<td>Very easily or after a scenario, go to 4.1.1. Happens after a high load or not very often, go to 4.5. If you cannot reproduce the crash at all, or you can reproduce it but only in release mode with no logs, contact Support.</td>
</tr>
<tr>
<td>4.1.1 Where does the crash occur?</td>
<td>ema.c or ra.c files, go to 4.3. pvt_pst directory, go to 4.4. Other, go to 4.2.</td>
</tr>
<tr>
<td>4.2 Stack crashes when calling an API.</td>
<td>Yes, go to 4.2.1. No, go to 4.5.</td>
</tr>
<tr>
<td>4.2.1 Did the API require passing a structure or a buffer (as in certain get-parameter functions)?</td>
<td>Yes, go to 4.2.1.1. No, go to 4.2.2.</td>
</tr>
<tr>
<td>4.2.1.1 Make sure to initialize the structure, set string members with valid buffers, and set length members to correct lengths. Also try providing longer buffers.</td>
<td>Did not help, go to 4.2.2. Crash no longer occurs, done.</td>
</tr>
</tbody>
</table>
### Troubleshooting

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| **4.2.2** | Try to follow the callback stack and see if the crash occurred when accessing handles in the called API. The handles seem to be corrupted or empty, go to **4.2.3**. If the handles seem to be OK, go to `<log place 1>`.
| **4.2.3** | Make sure the handles were not released already. If they were not or you cannot tell, go to `<log place 1>`.
| **4.3** | If the Stack crashes in `emaGetApplicationHandle()` or `emaGetInstance()`, it is usually because the handles were already released. If an API caused the crash, go to **4.2.3**. Otherwise, use `<log place 2>`, go to **4.6**.
| **4.4** | This sometimes happens when PVT nodes are accessed that were already freed. If it happened when passing a PVT node to an API, try printing the node using `pvtPrintStd()` to log handle (-1). If the function returns an error or crashes, the node was probably freed. If the root was created at the application level, you can check the VT logs to see if it was deleted and from where. If this does not help, or the node prints correctly, use `<log place 3>`, go to **4.6**.
| **4.5** | Make sure no resource problem occurs before the crash (see section 2) by looking for previous errors in the Log. Not a resource problem, use `<log place 4>`, go to **4.6**.
| **4.6** | Does the call stack include timer functions? Yes, use `<log place 5>`.
Log place 1:
Simple scenario crash log:
CMAPI, CMAPICB, Transport, TPKTCHAN, RAS, UDPCHAN, EMA, RA, VT.

Log place 2:
Handle crash log:
CMAPI, CMAPICB, Transport, RAS, EMA, RA.

Log place 3:
PVT crash log:
CMAPI, CMAPICB, TPKTCHAN, RAS, UDPCHAN, PVT.

Log place 4:
Timer log
TIMER, ETIMER

Log place 5:
High load/multithreaded log:
RA, EMA, CMAPI, CMAPICB.
5. DEADLOCK IN A MULTI-THREADED APPLICATION

Deadlocks are caused when two threads lock the same two objects but in reversed order, so that neither of them can continue. Symptoms are either that the Stack "hangs", or that the two threads are stuck.

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Make sure you compiled the Stack with multi-threading support, and that the OS you are using is supported with multi-threading (see differences between OS versions, below).</td>
<td>If everything seems to be OK, go to 5.2.</td>
</tr>
<tr>
<td>5.2 Send Support the call stack and the line in which both threads are stuck, and a log file.</td>
<td>Try &lt;logs place 5&gt;. If the logs are too big, or the Log prevents the deadlock, try &lt;log place 6&gt;. If this is still too big or prevents the deadlock, try &lt;log place 7&gt;. If any form of log prevents the deadlock, just send the call stacks.</td>
</tr>
</tbody>
</table>

**Log place 5:**  
EMA, CMAPI, CMAPICB, Transport, TIMER, ETIMER.

**Log place 6:**  
EMA, CMAPI, CMAPICB.

**Log place 7:**  
EMA.
6. MESSAGE RECEIVED BY OTHER SIDE IS DIFFERENT THAN WHAT APPEARS IN TPKTCHAN OR UDPCHAN

Messages appearing in TPKTCHAN and UDPCHAN might look different than what network sniffers see. This may be caused by one of the following.

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 First, use TPKTWIRE or UDPWIRE as additional log sources to make sure that the information in TPKTCHAN and TPKTWIRE (or UDPCHAN and UDPWIRE, depending on the problem) is identical.</td>
<td>Yes, the messages seem to be identical—Go to 1.7. No, they are different—Go to 6.1.1.</td>
</tr>
<tr>
<td>6.1.1 The difference between the two sources indicates that your application is modifying the message in the Stack hooks before the Stack sends the messages to the network.</td>
<td>Check the code in the implemented hooks.</td>
</tr>
</tbody>
</table>
7. RECEIVED MESSAGE DIFFERENT THAN WHAT APPEARS TO SNIFFER

Messages appearing in TPKTCHAN and UDPCHAN might look different than what network sniffers see. This may be caused by one of the following.

<table>
<thead>
<tr>
<th>Question</th>
<th>Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 First, use TPKTWIRE or UDPWIRE as additional log sources to make sure that the information in TPKTCHAN and TPKTWIRE (or UDPCHAN and UDPWIRE, depending on the problem) is identical.</td>
<td>Yes, the messages seem to be identical—Go to 7.2. No, they are different—Go to 7.1.1.</td>
</tr>
<tr>
<td>7.1.1 The difference between the two sources indicates that your application is modifying the message in the Stack hooks before the Stack begins to handle the messages.</td>
<td>Check the code in your implemented hooks.</td>
</tr>
<tr>
<td>7.2 Does the sniffer show multiple information elements from Q.931 fields?</td>
<td>Yes, this is the problem—Go to 7.2.1. No, something else is wrong—Go to 7.3.</td>
</tr>
<tr>
<td>7.2.1 The H.323 standard states that multiple information elements in Q.931 should not be handled, and the Stack ignores the multiple fields.</td>
<td>If you need a capability to support multiple IEs in Q.931 message, you can obtain the RADVISION SDK package and use it to implement this feature in your application. Please contact a RADVISION Sales Representative for this.</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 7.3 | Are there any PERERR warning messages in the Log just before the printout of the message, indicating an unknown extension? | Yes—Go to 7.3.1.  
No—Go to I.8 |
| 7.3.1 | Possibly your Stack version cannot handle the fields sent by the remote endpoint due to differences in the standard versions between endpoints. | This should not cause problems.  
In case of doubt, go to I.8. |
I. INFORMATION NEEDED BY CUSTOMER SUPPORT

The following list indicates the types of information that are needed by RADVISION Customer Support for handling the problems discussed on the previous pages. Please collect as much of this information as possible before contacting Customer Support.

Whenever you ask for support, please indicate the route you have taken in answering the questions on the previous pages and the scenario you were executing when the problem occurred. Note that using lower debug levels will reduce the size of the Log, but information needed to understand the problem more quickly might be left out. Optional log sources might help Customer Support find the problem more quickly, but will increase the size of the log file.

Samples of log settings described below can be found in the Add-ons directory of the Stack distribution, under Sample Configuration Files.

Problem:

I.1 Unspecified

- config.val file you are using
- Log with (debug level 1/2):
  * = ERROR EXCEPTION WARNING
  WATCHDOG
  CONFIG
  CMAPI
  CMAPICB
  UDPCHAN
  TPKTCHAN
- Sample file: rvtele-I1-Unspecified.ini
Troubleshooting

I.2 RAS

- config.val file you are using
- Log with (debug level 1/2):
  *=ERROR EXCEPTION WARNING
  WATCHDOG
  CMAPI
  CMAPICB
  RAS
  UDPCHAN
  TPKTCHAN
- Optional: EMA, CONFIG
- Sample file: rvtele-I2-RAS.ini

I.3 Regular PVT leak

- config.val file you are using
- Log with (debug level 1/2):
  *=ERROR EXCEPTION WARNING
  WATCHDOG
  CMAPI
  CMAPICB
  VT
- This log must be kept from the beginning of the execution of the Stack. Try to run short scenarios to have smaller log files for it.
- Sample file: rvtele-I3-RegularPVTLeak.ini
Troubleshooting

I.4 Transport

- config.val file you are using
- Log with (debug level 1/2):
  *=ERROR EXCEPTION WARNING
  WATCHDOG
  TIMEPOOL
  CMAPI
  CMAPI CB
  Transport
  UDPCHAN
  TPKTCHAN

- Optional: AnnexE if you are using H.323/AnnexE calls
- Optional: EMA
- Sample file: rvtele-I4-Transport.ini

I.5 Timer

- config.val file you are using
- Log with (debug level 1/2):
  *=ERROR EXCEPTION WARNING
  WATCHDOG
  CMAPI
  CMAPI CB
  UDPCHAN
  TPKTCHAN
  TIMER
  TIMEPOOL
  SELECT

- Make sure to run with a Watchdog interval of no more than 5 minutes. For short scenarios, use a Watchdog interval of no more than 30 seconds.
- Sample file: rvtele-I5-Timer.ini
I.6 Application timer

- config.val file you are using
- A description of the way you use ntimer functions and for what reason.
- Log with (debug level 1/2):
  * = ERROR  EXCEPTION  WARNING
  WATCHDOG
  MTI
  CMAPI
  CMAPICB
  UDPCHAN
  TPKTCB

- Sample file: rvtele-I6-ApplicationTimer.ini

I.7 Call association table

- config.val file you are using
- Log with (debug level 2 - 1 is not enough in this case):
  * = ERROR  EXCEPTION  WARNING
  WATCHDOG
  CAT
  CMAPI
  CMAPICB
  UDPCHAN
  TPKTCB

- Optional: RAS, Transport
- Sample file: rvtele-I7-CallAssociationTable.ini
I.8 ASN.1 or Q931 Encoding/Decoding Problem

- config.val file you are using
- Log with debug level 2 (debug level 1 is not enough in this case):
  *=ERROR EXCEPTION WARNING
  PER
  PERERR
  Q931
  Q931ERR
  UDPWIRE
  TPKTWIRE
- Sample file: rvtele-I8-EncodingDecoding.ini

Older H.323 Stack Versions

If you are using an older Stack version, you can still use the above information, but you should be aware of the following differences.

- Version 4 and earlier versions do not include a Watchdog module. This means that there is no Watchdog API available in these versions and there is no WATCHDOG log source to view. The Watchdog module is available from version 4.1.

- Version 4 and earlier versions do not include the ability to log specific types of messages from all sources. This means that using "*=ERROR EXCEPTION WARNING" will not work. We suggest that you add the following to all logs:
  PERERR, Q931ERR, CMERR, RA
  Note that these warnings and errors will be displayed in version 4.0 only. From version 4.1, the application can choose to log using "*=ERROR EXCEPTION WARNING".

- Version 4.1 and earlier versions do not include the Log sources TPKTWIRE and UDPWIRE, making it more difficult to verify whether the application has changed incoming or outgoing messages between the Stack and the network. This can be verified only by using a network sniffer for these versions.

- From version 4.2, the application can choose to log by using TPKTWIRE and UDPWIRE to get more information about modifications made by the application in hook callbacks between the Stack and the network.
- Version 3 and earlier versions do not include an EMA module. This means that troubleshooting by using EMA log source is not available for these versions. From version 4, the application can choose to log by using EMA to help the RADVISION Customer Support team find the problem.

- Version 4 and earlier versions do not include the ms APIs for Windows applications.

- Version 4 and earlier versions use msg.conf or the ms APIs for all operating systems other than Windows. Please make sure to use an appropriate log control mechanism for those versions.

- Version 4 and earlier versions do not include the ability to use several cyclic log files or the ability to change the name of the log file itself. This ability was added in version 4.1.

- Version 4.1 and earlier versions can cause VT warnings in normal call connection and disconnection scenarios. Please ignore such warnings in the Log.
A Denial of Service (DoS) attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service. Examples include:

- Attempts to "flood" a network, thereby preventing legitimate network traffic
- Attempts to disrupt connections between two machines, thereby preventing access to a service
- Attempts to prevent a particular individual from accessing a service
- Attempts to disrupt service to a specific system or person.

Not all service outages, even those that result from malicious activity, are necessarily DoS attacks. Other types of attack may include a denial of service as a component, but the denial of service may be part of a larger attack. ¹

In the context of H.323, DoS attacks are often viewed as external messages that have been engineered to cause an H.323 service to crash or to loose resources by taking advantage of unprotected code areas in the H.323 service. An H.323 service can be separated into two distinct parts: The RADVISION H.323 Protocol Toolkit, and the application that is written on top of it. The RADVISION H.323 Protocol Toolkit is constantly enhanced to offer the best possible protection against DoS attacks. It identifies most DoS attacks, ignores them, and clears up any used resources that might otherwise be lost. It does so through various mechanisms, including error checking, message sanity checks, and internal timeouts on events.

Although the Stack itself gives the application greater stability and protection from DoS attacks, the application must also protect itself. It must protect itself mainly from the use of limit values, including strings that reach their possible standard limit, and from very long aliases.

The sections below contain a list of Stack API functions that should be used with extra care for better protection against possible DoS attacks, and to make your application more stable.

**Note** The code examples in this chapter are based on the types of version 4.1 and above. To use them in version 4.0, you should replace RvChar with char, RvInt32 with INT32, etc.

¹ Source: CERT (http://www.cert.org/tech_tips/denial_of_service.html)
**ALIASES**

The most important aspect of the application involves dealing properly with aliases. Aliases can be up to 512 bytes long. Since the Stack always adds a NULL termination, aliases may actually need 513 bytes. When Stack API functions return a value of the type cmAlias as a parameter, the application is responsible for supplying the buffer for the string field inside the cmAlias.

The following three API functions in the Stack need to be handled properly in this regard:

```c

1. cmVi2Alias() for the parameters:
   cmParamSourceAddress, cmParamDestinationAddress, cmParamDestExtraCallInfo,
   cmParamCalledPartyNumber, cmParamCalledPartySubAddress, cmParamCallingPartyNumber,
   cmParamCallingPartySubAddress, cmParamExtension, cmParamAlternativeAliasAddress,
   cmParamAlternativeDestExtraCallInfo, cmParamAlternativeExtension, cmParamConnectedAddress

2. cmRASGetParam() for the parameters:
   cmRASParamGatekeeperID, cmRASParamEndpointAlias, cmRASParamTerminalAlias,
   cmRASParamEndpointID, cmRASParamDestInfo, cmRASParamSrcInfo,
   cmRASParamDestExtraCallInfo, cmRASParamRejectedAlias, cmRASParamSourceInfo,
   cmRASParamLocal436TerminalAlias, cmRASParamExtension.
```

There are two different options for dealing with the above functions:

1. Supply a string buffer that is 513 bytes long to avoid any possible buffer overruns that might lead to crashes later on.

This may be done using the following code, as in this example:
In the example above, the application sets the string field inside the sourceAlias variable to a string buffer that is 513 bytes long. Some applications would consider this a waste of memory (in most cases). For this reason, for RADVISION H.323 Protocol Stack version 4.2 and later, an alternate option exists:

2. Applications using RADVISION Protocol Stack version 4.2 and later can use shortened string buffers when dealing with aliases. To do so, the application must set the configuration parameter called system.checkParamLength. This causes the Stack to look at the length field inside of all cmAlias parameters that are passed as output parameters to the three API functions described above. In older applications, you should search for all calls to the above functions and modify your code accordingly.
For example:

```c
int cmEvNevCall()
{
    IN HAPPH app,
    IN XCALL hCall,
    OUT YHAPFCALL ypCall)
    
    /* Deal with incoming calls. See who is the source of this call */
    cmAlias sourceAlias;
    RPChar src[100]; /* Deal only with aliases that are 100 bytes long */
    IN int status;
    Rlnt32 aliasLen;

    /* Make sure the return value is NULL if you encounter trouble */
    ypCall = NULL;

    /* See what is the source address */
    sourceAlias.length = (lnt32)(sizeof(src));
    status = cmCallGetParam
    (hCall, cAPhAPSourceAddress, 0, aliasLen, (char*)sourceAlias);
    if ((status == 0) && (sourceAlias.length > (lnt32)(sizeof(src)))).
    {
        /* The alias received is more than 100 bytes long. In this case, you can
            choose to give a larger buffer, only deal with the 100 bytes you have, or
            drop this call. In our example here, it was decided to drop the call. */
        cmCallDrop(hCall);
        return 0;
    }
}
```

**Note** In the example above, the application indicated the buffer length in sourceAlias.length. After the call to cmCallGetParam() returned, it checked the value of sourceAlias.length again to see the real length of the alias it received.

**OBJECT IDENTIFIERS**

When dealing with object identifiers, the Stack parses the object identifiers into readable strings. These strings are not always friendly to the application and might have been engineered by a malicious remote endpoint to cause crashes in your application. Take care to compare and check the values of such incoming object identifiers in your code.
STRINGS

Strings received from the network inside aliases, display parameters, or any other component dealing with readable strings, might contain the ‘%’ character. For this reason, never pass the strings you receive from the Stack (if they are parts of network messages) to the function printf() or similar ANSI C functions as the first parameter. Doing so might cause the function to crash, as illustrated by the following example:

```c
{  
  char badString[100];
  strcpy(badString, "this string contains %p %s %d signs to cause your code to crash");
  printf(badString); 
}
```

You should do something like the following:

```c
{  
  char badString[100];
  strcpy(badString, "this string contains %p %s %d signs to cause your code to crash");
  printf("%s", badString);
}
```