

WHITE PAPERS from the files of Networking Unlimited, Inc.

WebQuery@NetworkingUnlimited.com

14 Dogwood Lane, Tenafly, NJ 07670

Phone: +1 201 568-7810

FAX: +1 201 568-7269

Automated Analysis of Cisco Log Files

Copyright © 1999, Networking Unlimited, Inc. All Rights Reserved.

Tremendous amounts of useful operations data and warnings of pending failures are available in the router logs. The challenge is that as the network gets larger, so do the number of entries in the logs, which can quickly grow to unmanageable size. Automating the analysis of router logs is essential to allow using the router logs as a proactive network management tool. Many organizations fail to take full advantage of the available information because of the high initial cost of programming around the various inconsistencies in the way various events are reported, the frequency with which individual entries are delayed, duplicated or missing, and the need to customize the software to match their network configuration. This paper looks at some of the techniques used by Networking Unlimited, Inc to improve the accuracy of automated log analysis and make it a cost-effective tool for network management and improving network reliability.

Table of Contents

[Overview](#)

[Preprocessing Input](#)

[Coding Examples:](#)

[Extracting Interesting Events](#)

[Monitoring a Single Line](#)

[Other Useful Techniques](#)

[Application Example: a Frame Relay Network with ISDN Backup](#)

[Bottom Line](#)

Overview

Cisco routers can provide an immense quantity of real time status information to support network management simply by enabling the system logging facility. Every state transition of every line can be recorded, along with call statistics, router configuration changes, software errors, environmental warnings, IOS reloads, and more. This level of detail can provide valuable insight into network operation that goes far beyond its normal use as a tool for resolving the cause of network failures. Many classes of problems, such as weak links which fail intermittently for brief periods, will show up in the syslog long before they are noticeable to users.

Given the valuable information and operational insight available from the system logs, detailed analysis of syslog data should be a routine part of every network administrators job. Yet in real life we find that this resource is frequently ignored. simply because of the difficulty of dealing with the huge quantity of raw data. Any single physical event can easily generate a dozen or more log entries. Even a simple drop on a leased line will generate a link layer down notice and a physical layer down notice from the routers at each end of the line, followed by the up notices for both layers from both routers when the line returns to normal. If the link is frame relay, there will also be log entries for the associated DLCIs going inactive or deleted (and later returning to the active state). If dial backup is in place, there will also be physical and link up (and then down) notices for the backup medium and if the backup is ISDN, there will be ISDN connect and disconnect statistics. Add them all up and a single link hit can generate twenty or more log entries

If the network incident happens to involve a shared resource, such as a frame relay link supporting dozens of PVCs, the result can be page after page of log data simply because of one event. Even if there are no failures to report, routine testing of dial backup links will generate log entries which still must be scanned to ensure that they all really are test results and not a problem which just happened to occur during test periods. In a network with hundreds of routers, the logs can easily contain thousands of lines of entries for each day and if there are unstable lines, the thousands can expand to tens of thousands. As a result, many network administrators only look at the log files when debugging an already detected problem, and do not take advantage of the proactive network management possible with routine log analysis. This prevents them from detecting the early warning signs of impending problems such as brief outages, unexplained dial backup calls, and patterns in the failures occurring.

Clearly, analysis of syslog data is a natural application for automated processing. The problem is that while a cursory glance at a log shows obvious patterns which should be easily handled by simple software, there are a number of properties inherent to the nature of router log files which interfere with automated analysis. Simple scripts to analyze the data tend to fail because despite the quantity of data reported, the log files are frequently incomplete. This is a natural side effect of the nature of remote system logs. The log events are sent over the network using the user datagram protocol (UDP) which provides only "best-efforts" data packet delivery. If for any reason a log entry packet fails to get delivered, no attempt is made to recover from the error and that log entry will never be recorded. If the event being reported is the loss of the link which must carry the syslog entry packet, the message that the link has gone down will almost never be received. Similarly, the messages associated with a link coming back up or dial backup being connected are also frequently lost, depending upon how long it takes for the routing protocols to converge on the new routes.

Nor are missing records the only data corruption which must be countered. Packets may also be delivered out of order, they may be delayed varying amounts so that the timestamps added by the destination syslog daemon are inconsistent, and sometimes the same packet may be delivered multiple times.

Adding to the challenge is the inconsistency in the formatting and sequencing of reports for various common events and facilities. For example, the sequence of reports for a frame relay link going down includes notification of DLCI status changes on the PVC, so that even though the physical link is up and the link layer protocol is up on the interface, the ability to communicate is down. Yet on a leased line, the same physical and link layer reports are not only all that is required, but also all that is available. ATM, X.25, asynchronous dial, Ethernet, all follow different sequences which must be interpreted in the context of the specific link.

What is reported and how also varies between IOS releases and may change based on router configuration settings. But the most significant restriction is the device-centric nature of the syslog data. The impact of a link going down depends upon the network architecture and remaining connectivity, which is not part of the log file but is essential to interpreting the results. There is nothing in the log file to even indicate if a pair of link down reports received from two different routers means a link between them is down or that two independent links are down. As a result, attempting to simply apply an off-the-shelf analysis tool is almost certain to yield disappointing results.

Networking Unlimited has found that while each network environment is unique, requiring investment in customized analysis software, the results are well worth the effort. Particularly in networks designed with multiple levels of redundancy for high reliability, the routine analysis of day to day operation is essential to keep track of what is working and what is failing. The key is to detect and correct problems before they become noticeable rather than waiting until users start complaining. In the next few sections, we will look at the specific programming algorithms and analysis techniques that Network Unlimited uses to automate log analysis processing. Those who want skip the coding

details and go directly to the bottom line results should click [here](#).

Preprocessing the Input

The first step in processing the log data is to clean it up. In a typical syslog, the format of each entry is a time stamp provided by the local syslog daemon marking when the entry was received for log processing, the source of the message (usually an IP address or system name), then the message itself. The message body of Cisco syslog messages consists of a message sequence number (maintained independently by each router sending syslog messages), an optional timestamp provided by the router, the message class, severity, and name, and finally, a text explanation. Some typical syslog entries might look like:

```
Sep 1 00:00:36 10.101.13.13 1687: %LINK-3-UPDOWN:  
Interface BRI0: B-Channel 1, changed state to up
```

```
Sep 1 08:51:13 10.104.60.5 1739: 3:40:21: %FR-5-  
DLCICHANGE: Interface Serial0 - DLCI 22 state changed to  
INACTIVE
```

```
Sep 1 18:15:41 10.104.56.13 310: %LINEPROTO-5-UPDOWN:  
Line protocol on Interface Serial0.21, changed state to  
down
```

While nothing can be done to recreate missing data, it can be detected by checking the sequence numbers. But before we can check the sequence numbers, we must convert the source address into a canonical address for each device because the source address as sent by the router will be the address associated with the interface used to send it, which in turn will vary depending upon the best route available at the time of generation. Once we have a standard name associated with the entry, we can then check that the sequence numbers make sense. By maintaining a look ahead buffer of log entries, we can sort out the log entries from each device by sequence

number and if configured, by source timestamp, so that log entries which were received out of order can be processed correctly.

Of course any time the order of log entries is modified, the syslog timestamps may also need to be adjusted to ensure that time never goes backwards. We must also guard against router reboots, as a reboot causes the sequence numbers to reset and start counting up from zero again. If we just blindly took the lowest sequence number available, we might never process the events immediately preceding the crash. This can be handled by limiting the look ahead buffer size by the timestamps rather than by a fixed number of entries. That way, we always limit our look ahead horizon for any router to be shorter than the time required for that router to reboot without restricting our look ahead window to fit the fastest router in the network. We can also detect missing entries (because sequence numbers were skipped), discard duplicate entries (a brief routing loop can cause hundreds of duplicates), and ensure that the look ahead buffer is adequately sized (if the sequence numbers go down for a device, yet do not go low enough to indicate a reboot, we probably have delayed packets which were too delayed to fix).

A less obvious benefit of preprocessing the raw log data is that it simplifies the use of multiple logs in an analysis. For example, if each router logs to both a local system (connected via LAN) and to a syslog server in the network operations center, the two logs can be combined to greatly reduce the number of missing event entries without having to deal with the duplication of entries in the main analysis program.

Not only must we "clean up" the log data before processing it for events, we must also detect more general events, such as a router rebooting, which can change the state of many lines on the router without generating any reports. For example, if a router loses power, it will not generate any link down messages, but because it rebooted, we know that all links did go down and may or may not have come back up.

The *perl* examples which follow all depend upon the log being read using a subroutine *cookedlogread* which parses the next useful line in the log into an array consisting of a normalized source ID, normalized time stamp, sequence number, the original text of the log message, and the log message without timestamps, sequence numbers or other already parsed data (normally just everything after the % sign). The *cookedlogread* subroutine will read ahead into the log file to put out-of-order entries for each source ID back into sequence number order, verify that time is monotonically increasing (correcting for daylight savings time transitions as necessary if the log uses local time), report

if the router appears to have been rebooted (sequence numbers restarting at a low value), detect and report time gaps in the log file (assuming that marking is enabled), detect and report missing entries (assuming that the receiving syslog daemon is not configured to filter out lower priority packets), and otherwise attempts to ensure that the data being processed is as clean as possible.

Coding Example: Keeping a Watch for "Interesting Events"

A useful program which is easy to implement is keeping a watch on the log file for interesting events. Networking Unlimited always runs this filter on any log file being analyzed to ensure that we are considering all useful events in our normal analysis. Typically, an interesting event is defined as any event other than those associated with a line going up or down. This simplifies detection of problems which just happen to occur during a period of network instability where the log is being filled with link up and down reports. Logically, the program simply implements a reverse grep function, printing any log entries which are recognized to be from a Cisco router that do not match any patterns defined as uninteresting entries. For example, in the code segment below used for a frame relay network with ISDN backup, we ignore link up/down, DLCI state changes, ISDN state changes, and ISDN call statistics.

```
unless ( ($line =~ /\%(\S+):\s+(.*)$/ ) || ($line =~ /(-
Process|-Traceback)=\s+(.*)$/ ) )
{
    if ($line =~ /Cisco Internetwork Operating System
Software/)
    {
        #ignore bootup banner messages
        $msgtypes{"Bootup Banner"}++;
        $ignoretil{$logline[0]}=$logline[2]+4;
        next LOGLINE
    }
    # Remainder of boot up banner recognition and other spec
ial cases go here
    # Only get to here if could not parse the line
    print "*** $logline[3]\n";
    $msgtypes{"!!! unknown !!!"}++;
    $logerrors++;
    next LOGLINE;
}
$msgtypes{$1}++;
unless (($1 eq "LINEPROTO-5-UPDOWN") or ($1 eq "LINK-3-
UPDOWN") or ($1 eq "FR-5-DLCICCHANGE")
        or ($1 eq "ISDN-6-DISCONNECT") or ($1 eq "ISDN-6-
CONNECT") or ($1 eq "LINK-5-CHANGED")
        or ($1 eq "ISDN-6-LAYER2DOWN") or ($1 eq "ISDN-6-
LAYER2UP" ) )
{
    # Interesting line... print it
    $timestamp = gmtime($logline[1]);
```

```
    printf "-->
%12s %s %8d %s\n", $logline[0], $timestamp, $logline[2], $log
line[4];
}
```

Even in this simple example, the preprocessing of log entries simplifies and enhances the analysis. For example, we can use the canonical name of the device reporting an event to key into associative arrays that keep track of the state of each device. The first "if" clause shows how we keep track of the detection of a router bootup banner to allow faster skipping of the remainder of the banner entries on subsequent lines (which is not shown in this excerpt). Note that we can not depend upon the presence of a bootup banner to detect a router reboot, as that message will only be logged if there are no delays in links coming on line and routing adjacencies being established during the reboot process, which usually translates to the only reboot messages in the log are those from routers on the same LAN as the syslog server.

Coding Example: Tracking Status of a Single Link

A major step up in complexity is using the event information reported in the log to track the state of a line. In this example, we monitor the availability of a single critical link and its backup, allowing network management to demonstrate to upper management the impact of reliability enhancements. We use the entries in the log file to drive a state machine describing the state of the line so we can maintain a continuous record of link operation. First, of course, we must sort the log entries by sequence number to restore their temporal relationships. For example, we want a link down message delayed in delivery until after the link up message following a brief service outage to be recognized as a link down that has been cleared rather than a new failure still in effect. However, it is not enough to simply restore correct ordering and eliminate duplicates.

Consider the sequence of log entries for a leased line between two serial ports. When a link comes up, it is first reported up at the physical layer with "%LINK-3-UPDOWN: Interface Serial0, changed state to up." Then, when the link protocol becomes active, the availability of a working link is reported with "%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0, changed state to up." If the link were frame relay rather than a leased line, we would also need a report that the DLCI was ACTIVE before we could declare the link to be up. At first glance, tracking the state of the link is simply a matter of parsing the lines to recognize the router, the interface on the router, and the action occurring on that interface.

The problem is that log entries are frequently missing. As a result, we often need to infer the correct state based on what we have heard rather than blindly stepping from state to state in response to individual log entries. For example, if we receive the sequence "link up" followed by "line protocol down" we know that at some point between the "link up" and the "line protocol down" was a "line protocol up" that did not get recorded. If we're lucky, we may have a "line protocol up" from the other end of the line, and simply assume that both ends of the line came up at about the same time. The key is to not only maintain state information, but also keep track of their history so that missed events can be inserted at the appropriate time when they are finally discovered.

Note that what can be safely inferred will also depend upon the type of link involved. For example, a line protocol up report from one end of a link implies both ends are up if it is a point-to-point leased line, but says nothing about the ability of a frame relay or ATM link to carry traffic (other than that it was definitely down before the event occurred).

Inferring the correct link state from the log history, while essential to correct for missing events, does reduce the utility of log analysis as a real time monitor, as we can expect to change our concept of the current state from time to time when new information arrives and we back track to make corrections for missed events. What we can do is use the implicit dependencies between various syslog messages to force consistency whenever possible. Continuing with our example of a simple leased line, we know that any time we see a "link down" report, the line protocol is also down, whether or not we detected it. Other useful dependencies include: "protocol up" or "protocol down" from one end of a leased line implies the equivalent state at the other end, "link down" implies "line protocol down", "line protocol up" implies "link up;" and "DLCI ACTIVE" implies "line protocol up" and "link up."

The code segment referenced below applies these strategies to monitor the status of a leased line on a router that is backed up by an automatically dialed ISDN line (configured as a "usually down leased line") on a second router. The line is the access line to a critical service provider for one of our clients. The link itself is controlled by the service provider and we wanted to monitor the quality of service actually being provided. Because both routers are logging to a local system, lost packets are rare. Not included in the code extract is the code in *cookedlogread* to handle potential down time when a router reboots, parameter and variable initialization, and summary statistics processing.

[Code segment reference](#)

The output of the analysis, a sanitized sample of which follows, is clearly far more readable and consequently more useful than the 120,000 lines of log file analyzed to generate it. The analysis requires about six minutes to run on a 233 MHz Pentium II.

```
Logging starts at Tue May 11 00:00:56 1999
-> UP <-
  Log starts Tue May 11 00:00:56 1999. Assuming leased line up and ISDN down
-> note <-
  RouterTwo rebooted between Wed May 12 20:25:40 1999 and Wed May 12 20:36:39 1999
  --
> ISDN protocol state changed from down to down on leased line
-> note <-
  RouterTwo rebooted between Wed May 12 20:38:18 1999 and Wed May 12 20:51:34 1999
  --
> ISDN protocol state changed from down to down on leased line
-> DOWN <-
  RouterOne Fri May 14 21:31:17 1999 LINEPROTO-5-UPDOWN down
-> ISDN <-
  RouterTwo Fri May 14 22:14:28 1999 LINEPROTO-5-UPDOWN up
-> UP <-
  RouterOne Sat May 15 02:30:44 1999 LINEPROTO-5-UPDOWN up
-> ISDN <-
  RouterOne Sat May 15 02:53:02 1999 LINEPROTO-5-UPDOWN down
-> UP <-
  RouterOne Sat May 15 02:54:29 1999 LINEPROTO-5-UPDOWN up
- drop -
  RouterOne from Mon May 17 11:39:36 1999 to Mon May 17 11:39:46 1999
-> DOWN <-
  RouterOne Mon May 17 11:40:16 1999 LINEPROTO-5-UPDOWN down
- try -
  RouterOne from Mon May 17 11:43:33 1999 to Mon May 17 11:44:04 1999
- try -
  RouterOne from Mon May 17 11:45:19 1999 to Mon May 17 11:45:52 1999
- try -
  RouterOne from Mon May 17 11:46:36 1999 to Mon May 17 11:47:08 1999
-> UP <-
  RouterOne Mon May 17 11:48:27 1999 LINEPROTO-5-UPDOWN up
- drop -
  RouterOne from Mon May 17 11:49:38 1999 to Mon May 17 11:49:55 1999
- drop -
  RouterOne from Mon May 17 11:50:26 1999 to Mon May 17 11:50:41 1999
-> ISDN <-
  RouterOne Thu May 20 19:17:45 1999 LINEPROTO-5-
```

```

UPDOWN down
-> DOWN <-
    RouterTwo rebooted between Thu May 20 19:48:19 1999 a
nd Thu May 20 19:48:56 1999
***      RouterTwo Thu May 20 19:48:57 1999      161 %LINK
-5-
CHANGED: Interface Serial1/3, changed state to administra
tively down
-> note <-
    RouterTwo rebooted between Thu May 20 19:48:51 1999 a
nd Thu May 20 19:49:06 1999
-> note <-
    RouterTwo rebooted between Thu May 20 19:49:44 1999 a
nd Thu May 20 19:50:04 1999
-> DOWN <-
    RouterTwo Thu May 20 19:48:57 1999 LINEPROTO-5-
UPDOWN down
-> note <-
    RouterTwo rebooted between Thu May 20 19:50:03 1999 a
nd Thu May 20 19:50:18 1999
-> note <-
    RouterOne rebooted between Thu May 20 19:33:54 1999 a
nd Thu May 20 20:13:48 1999
--
> Line physical state changed from up to up on leased l
ine
- try -
    RouterOne from Thu May 20 20:18:51 1999 to Thu May 20
20:19:15 1999
--
> ISDN protocol state changed from down to down on leas
ed line
-> UP <-
    RouterOne Thu May 20 20:22:36 1999 LINEPROTO-5-
UPDOWN up

```

Logging ends at Thu May 20 23:59:54 1999

```

=====
==

```

PROCESSING SUMMARY

Log covers from Tue May 11 00:00:56 1999 to Thu May 20 23:59:54 1999
with 0 gaps of 23 minutes or longer detected.

```

    124,354 log file lines examined
    124,354 log file entries analyzed
    0 log file lines unrecognizable

```

Log covers a total of 9 days, 23 hours and 58.9 minutes.
Normal leased line operation totaled 9 days, 17 hours and 44.3 minutes, or 97.398%
Total time running on ISDN backup was 4 hours and 43.4 minutes, or 1.968%
Total down time detected was 91.3 minutes, or 0.634%
%
Overall Critical Service availability was 99.366%

Other Useful Techniques

One challenge is determining the state of all links at the start of an analysis run. Requiring the user to manually enter the state of every link before starting an analysis is a good way to guarantee that the tool will never be used. Ideally, we would like to couple the state information from a network management tool such as OpenView or Cisco Works to provide the starting status for each link. But aside from being difficult to extract, that information may not be consistent with reality at the time required, as there can be delays in detecting and reporting state changes. An alternate approach is to provide the ability to save state information at the end of an analysis and use that to start the next analysis.

The easiest approach is to deduce the initial state based on the first event or state indication detected. This can be remarkably accurate with longer sample times in environments that include daily activities such as dial link testing or configuration archiving. For example, in an analysis of frame relay with ISDN backup, a report of frame going down would be interpreted as an indication that the frame relay link started in the up state. Similarly, a report of ISDN going down strongly implies that ISDN was up to start, which in turn implies that frame relay is probably down. In the same way if the first report for a link has it coming up, it can usually be assumed that the link had been down. Hints can also be used from log entries generated by events other than link state changes. The trick is to determine what makes sense and then backtrack to the start of analysis to calculate up time and down time with the appropriate assumptions.

A similar approach must be taken at the end of the analysis run, when all timers and totals must be completed based on the time of the last log entry. At that point, a best guess for links which have no other state information available would be to assume that "permanent" links started and ended in the up state while "dial backup" links were unused. This would miss a dial backup link that was in use at the beginning and never changed state, but the longer the analysis period the less likely that nothing would have occurred to generate at least a hint indicating what link was in use.

ISDN call records provide an extra degree of back tracking which can be useful in a "call center" environment. Consider the situation where a remote site determines that frame relay has failed and initiates a call to the hub router. The log events generated by the remote are not recorded at the hub because they are generated before the ISDN link is up and available. At the hub, there are reports of the ISDN call coming in, but since the call could be from any remote site, there is no

way to determine which remote has dialed in. Using CHAP authentication and Caller ID will usually provide this information, but given the frequency of lost syslog reports, can not be depended upon. Sometimes we will not learn who called until the call completion record is found, which usually arrives from both ends (unless ISDN link failure forced the call down) and includes the total duration of the call. Backtracking can then be used to match up the call completion with the incoming call already detected, and statistics corrected as appropriate.

One problem with ISDN call records is that peer names may not be included in connection establishment records if authentication is delayed for any reason. One workaround is to use save the Caller ID information and wait until the disconnect record to determine the remote user name by matching the phone number to the name at that time. It is also possible to build a table mapping phone numbers to user names so that backtracking is only required if authentication is delayed on the first call from a site.

Tracking frame relay is fairly straightforward using the techniques already mentioned, although we will see that there are some potential pitfalls for the unwary. Most important is the inability to depend upon reports from the remote sites, as their log entries are frequently lost when lines go down. This can lead to strange results such as backup calls placed while the log entries still indicate that frame relay is fully functional (which, of course, it is not).

At the same time, since even local event reports can get lost, particularly if many links are changing state simultaneously, as will happen when a frame relay T1 with dozens of subinterfaces defined goes up or down and each transition of the T1 generates a link up/down, a line protocol up/down, and a series of DLCI state changes on every subinterface, we need the ability to "second guess" the log events when determining the state of links. Cisco actually makes this task significantly easier because the default behavior of event reporting is to use as a return address the address of the interface which is physically sending the packet. Normally, the source IP address used in the syslog entry will be that of the interface used to send the report. Unnumbered interfaces will use the address of the interface designated in the "ip unnumbered" command.

Any time we see a message from a link interface, we know that that link is up regardless of whether or not we have seen any link up messages. So if we receive an ISDN down report (or any other event, for that matter) from a remote system whose frame relay link we believe to be down, and that log entry is identified with the IP address

of the frame relay interface, we know that frame relay is actually up and we can correct the state information for that link accordingly. Similarly, if any event in the log came from the IP address of the ISDN link, we know that the remote system is running on ISDN backup, regardless of what we believe the state of the frame relay link to be and whether or not we detected the ISDN call connect.

If using IP unnumbered on any links, define a different loopback address for each mode of communications. That way, hinting will be able to pinpoint the actual mode of communications and determine which links are up to each device without consuming excessive IP address space. (If you use this approach, we also recommend assigning IP addresses to the loopback ports so that they can be aggregated and advertised as a single small subnet rather than multiple individual host addresses.) Using loopback addresses rather than a LAN interface address is essential when the network is expanded and a second router is installed at a site, as otherwise you could not determine what route was taken by the log event report from the router.

Since we convert the address used in the syslog event to a standard identifier we can use for processing, this information would normally be lost. What we do is define another entry in the array returned by "cookedlogread" which makes the source address hints visible to the main processing loop. Using these hints, we can then correct errors in state information. However, hints must be used carefully, as under certain conditions, such as a frame relay DLCI going INACTIVE, links may continue to function for a short period of time after they are reported down. So a hint must not be used to override an event entry unless sufficient time has passed to stabilize the interface, thirty seconds is normally more than enough time.

The key is to retain the knowledge of what events and hints were detected and when, along with the knowledge of current state. Then when a discrepancy is detected, such as a hint which contradicts current state information, the appropriate action can be taken. For example, a hint indicating that frame relay is up should be ignored on any report indicating frame relay going down (particularly DLCI state changes), and taken with a large grain of salt on any event within a few tens of seconds of the link being reported down. However, these reports should not be simply ignored, as they provide a hint as to when the link may have come back should events in the future indicate that a link up transition was missed (in other words, even though we ignored the hint, it may turn out to have been valid). When performing after the fact analysis of performance (as opposed to real

time tracking of link states), significant back tracking can be used to keep the performance statistics as accurate as possible.

Analysis in Action--Tracking Status of Multiple Links with DDR Backup

As an example of just how much interpretation is possible, let's look at what can be done to keep track of connectivity in a moderate sized hub and spokes network using frame relay with ISDN backup. Sanitized output from an analysis performed using many of the techniques discussed above is provided in [Demo Output File](#). While this is a large file (almost 3500 lines), it is far more manageable than the 575 pages of raw log file which it summarizes. The data is also in a form far more amenable to further analysis. Chronological order makes it easy to spot events which affect multiple sites such as a frame relay switch going down. Sorting the data by remote site name (the first entry on each line) makes it easier to spot problems with any single site which might otherwise be lost in the noise, as can be seen from the file [Demo Output File Sorted by Site](#).

Starting at the very beginning of the log are two examples of ISDN call reporting. The first, for remote 311, is an ISDN call detected while frame relay was up (what we call an "idle" call). Though when it appears in isolation, this extremely brief call is a mystery, if we *grep* the report for "remote311" a pattern emerges. Since a call around 6:00 AM would be considered a test call and not be reported, we can immediately recognize that the problem here is that the ISDN test for this site has been mis-programmed and is calling every six hours rather than every 24.

```
remote311 - isdn - Mon Sep 21 00:00:27 1998: Idle ISDN
call to hqLANrtr3 dropped after 0:00:10, call lasted
0:00:14
remote311 - isdn - Mon Sep 21 12:00:21 1998: Idle ISDN
call to hqLANrtr1 dropped after 0:00:02, call lasted
0:00:07
remote311 - isdn - Mon Sep 21 18:00:18 1998: Idle ISDN
call to hqLANrtr3 dropped after 0:00:04, call lasted
0:00:07
remote311 - isdn - Tue Sep 22 00:00:29 1998: Idle ISDN
call to hqLANrtr1 dropped after 0:00:07, call lasted
0:00:11
remote311 - isdn - Tue Sep 22 12:00:12 1998: Idle ISDN
call to hqLANrtr3 dropped after 0:00:04, call lasted
0:00:08
remote311 - isdn - Tue Sep 22 18:00:23 1998: Idle ISDN
call to hqLANrtr1 dropped after 0:00:01, call lasted
0:00:05
remote311 - isdn - Wed Sep 23 00:00:37 1998: Idle ISDN
call to hqLANrtr3 dropped after 0:00:07, call lasted
0:00:11
remote311 - isdn - Wed Sep 23 12:00:23 1998: Idle ISDN
```

```
call to hqLANrtr1 dropped after 0:00:03, call lasted
0:00:07
remote311 - isdn - Wed Sep 23 18:00:57 1998: Idle ISDN
call to hqLANrtr3 dropped after 0:00:16, call lasted
0:00:20
```

The second entry in the analysis report starts a sequence of entries for remote 017 indicating a much more serious problem. As can be seen from the first few entries below, this site is running on ISDN and the ISDN will not stay connected. As we continue to scan entries for this site, we see that the pattern is random with an occasional call lasting for hours. This is clearly a physical problem with the ISDN line (configuration errors almost always show strong calling patterns rather than random), and the problem was not detected until the ISDN line failed completely on the afternoon of Sep 24. In this case three days of down time could have been avoided by acting on the information in the logs rather than waiting for the user to complain of problems.

```
remote017 -> DOWN <- Mon Sep 21 00:00:54 1998: Dropped
ISDN call to hqLANrtr1 after 0:00:41, call lasted 0:03:59
remote017 -> ISDN <- Mon Sep 21 00:01:00 1998: Up on ISDN
to hqLANrtr1 after DOWN for 6 seconds
remote017 -> DOWN <- Mon Sep 21 00:06:39 1998: Dropped
ISDN call to hqLANrtr1 after 0:05:39, call lasted 0:05:43
remote017 -> ISDN <- Mon Sep 21 00:06:44 1998: Up on ISDN
to hqLANrtr1 after DOWN for 5 seconds
remote017 -> DOWN <- Mon Sep 21 00:14:38 1998: Dropped
ISDN call to hqLANrtr1 after 0:07:54, call lasted 0:07:58
remote017 -> ISDN <- Mon Sep 21 00:14:43 1998: Up on ISDN
to hqLANrtr1 after DOWN for 5 seconds
remote017 -> DOWN <- Mon Sep 21 00:18:37 1998: Dropped
ISDN call to hqLANrtr1 after 0:03:54, call lasted 0:03:58
remote017 -> ISDN <- Mon Sep 21 00:18:43 1998: Up on ISDN
to hqLANrtr1 after DOWN for 6 seconds
remote017 -> DOWN <- Mon Sep 21 00:22:37 1998: Dropped
ISDN call to hqLANrtr1 after 0:03:54, call lasted 0:03:59
.
.
.
remote017 -> DOWN <- Thu Sep 24 13:34:04 1998: Dropped
ISDN call to hqLANrtr1 after 0:03:58, call lasted 0:04:02
remote017 -> ISDN <- Thu Sep 24 15:36:12 1998: Up on ISDN
to hqLANrtr1 after DOWN for 2 hours and 2.1 minutes
remote017 -> DOWN <- Thu Sep 24 15:40:19 1998: Dropped
ISDN call to hqLANrtr1 after 0:04:07, call lasted 0:04:11
remote017 -> ISDN <- Thu Sep 24 16:22:10 1998: Up on ISDN
to hqLANrtr1 after DOWN for 41.9 minutes
remote017 -> DOWN <- Thu Sep 24 16:28:22 1998: Dropped
ISDN call to hqLANrtr1 after 0:06:12, call lasted 0:06:16
remote017 -> ISDN <- Thu Sep 24 16:33:48 1998: Up on ISDN
to hqLANrtr1 after DOWN for 5.4 minutes
remote017 -> DOWN <- Thu Sep 24 16:42:30 1998: Dropped
ISDN call to hqLANrtr1 after 0:08:42, call lasted 0:08:47
remote017 -> ISDN <- Sun Sep 27 10:33:08 1998: Up on ISDN
to hqLANrtr1 after DOWN for 65 hours and 50.6 minutes
```

Scanning down to 3:31 in the morning of the first day, we see another ISDN call, this time from remote 612. Again, in isolation this call is not

very meaningful, but a grep for "remote612" reveals the truth. As can be seen below, this call is one of several indicating a failing frame relay link. The "useless" idle ISDN calls are really backup calls for frame relay failures which were detected by the routing protocol but not severe enough to cause the line (or the PVC) to drop.

```
remote612 - isdn - Mon Sep 21 03:31:51 1998: Idle ISDN
call to hqLANrtr2 dropped after 0:03:10, call lasted
0:03:13
remote612 -> DOWN <- Tue Sep 22 13:40:22 1998:
Disconnected after at least 37 hours and 40.2 minutes on
frame to hqLANrtr3
remote612 -> ISDN <- Tue Sep 22 13:40:27 1998: Up on ISDN
to hqLANrtr2 after DOWN for 5 seconds
remote612 -> UP <- Tue Sep 22 13:43:23 1998: Up on frame
to hqLANrtr3 after on ISDN for 176 seconds

remote612 - ERR! - Tue Sep 22 13:44:34 1998: Frame up to
hqLANrtr3 71 seconds after already up
remote612 - isdn - Tue Sep 22 13:47:36 1998: Idle ISDN
call to hqLANrtr2 dropped after 0:04:13, call lasted
0:07:13
remote612 -> DOWN <- Tue Sep 22 22:04:34 1998:
Disconnected after 8 hours and 21.2 minutes on frame to
hqLANrtr3
remote612 -> ISDN <- Tue Sep 22 22:04:38 1998: Up on ISDN
to hqLANrtr2 after DOWN for 4 seconds
remote612 -> UP <- Tue Sep 22 22:05:34 1998: Up on frame
to hqLANrtr3 after on ISDN for 56 seconds
remote612 - isdn - Tue Sep 22 22:08:55 1998: Idle ISDN
call to hqLANrtr2 dropped after 0:03:20, call lasted
0:04:20
remote612 - isdn - Wed Sep 23 08:44:13 1998: Idle ISDN
call to hqLANrtr3 dropped after 0:03:35, call lasted
0:03:39
```

The importance of reporting inconsistent indications can be seen below in the sequence of events for remote 113. In this case, the frame link did drop at the remote site, forcing the ISDN link up, but the drop did not reach the syslog server at the core, nor did the core end of the frame relay link indicate any problems (the DLCI stayed up, an example of why we recommend against using "backup interface" to protect against frame relay failure). In this case, the four day long "idle" ISDN call was really an active ISDN call, but the frame relay failure was never reported. The necessary back tracking in this case was not performed, so the summary performance statistics would need to be manually adjusted.

```
remote113 - isdn - Thu Sep 24 09:56:52 1998: Idle ISDN
call to hqLANrtr2 dropped after 0:00:27, call lasted
0:00:33
remote113 - ERR! - Mon Sep 28 11:29:06 1998: Frame up to
hqLANrtr2 4 days and 23.2 hours after already up
remote113 - isdn - Mon Sep 28 11:32:15 1998: Idle ISDN
```

```
call to hqLANrtr2 dropped after 4d 01:34:51, call lasted
4d 01:35:04
remote113 -> DOWN <- Mon Sep 28 12:39:31 1998:
Disconnected after 5 days and 0.4 hours on frame to
hqLANrtr2
```

Another example of an abnormal frame hit at remote 520 starts at 8:24:51 on Sep 21. At this point, the hint that frame is up generated by the early morning ISDN call (which would have been reported over the frame relay link) has already established that the site is operational on frame relay. On the other hand, since we do not know when the link came up on frame relay because it was before the start of the log being analyzed, we only count the up time since the first time stamp in the log. The UP indication with two asterisks indicates that the status was determined by hints rather than by an explicit up report. While in isolation, this could be considered a normal dropped packet, the fact that the pattern repeats multiple times is suspicious, as it could mean that the ISDN link being brought up is incapable of reliably carrying traffic.

```
remote520 -> DOWN <- Mon Sep 21 08:24:51 1998:
Disconnected after at least 8 hours and 24.6 minutes on
frame to hqLANrtr1
remote520 -> ISDN <- Mon Sep 21 08:24:56 1998: Up on ISDN
to hqLANrtr3 after DOWN for 5 seconds
remote520 -> DOWN <- Mon Sep 21 08:27:20 1998: Dropped
ISDN call to hqLANrtr3 after 0:02:24, call lasted 0:02:28
remote520 ** UP <- Mon Sep 21 08:27:20 1998: Up on frame
to hqLANrtr1 after DOWN for 0 seconds
```

Later that morning, at 9:40:25 at remote544 we see an example of frame relay flapping after ISDN has kicked in. This is fairly common, as the frame relay line struggles to return to life. If desired, the analysis program could be modified to add hysteresis to the reporting (as was done in the single link coding example) so that a sequence of events like this would be reported as a single frame relay failure rather than multiple frame relay failures.

```
remote544 -> DOWN <- Mon Sep 21 09:40:25 1998:
Disconnected after at least 9 hours and 40.2 minutes on
frame to hqLANrtr2
remote544 -> ISDN <- Mon Sep 21 09:40:29 1998: Up on ISDN
to hqLANrtr1 after DOWN for 4 seconds
remote544 -> UP <- Mon Sep 21 09:40:39 1998: Up on frame
to hqLANrtr2 after on ISDN for 10 seconds
remote544 -> ISDN <- Mon Sep 21 09:40:41 1998: On ISDN to
hqLANrtr1 after 2 seconds on frame to hqLANrtr2
remote544 -> UP <- Mon Sep 21 09:40:59 1998: Up on frame
to hqLANrtr2 after on ISDN for 18 seconds
remote544 -> ISDN <- Mon Sep 21 09:41:01 1998: On ISDN to
hqLANrtr1 after 2 seconds on frame to hqLANrtr2
remote544 -> UP <- Mon Sep 21 09:42:01 1998: Up on frame
to hqLANrtr2 after on ISDN for 60 seconds
remote544 -> ISDN <- Mon Sep 21 09:42:18 1998: On ISDN to
hqLANrtr1 after 15 seconds on frame to hqLANrtr2
```

```

remote544 -> UP <- Mon Sep 21 09:42:35 1998: Up on frame
to hqLANrtr2 after on ISDN for 17 seconds
remote544 -> ISDN <- Mon Sep 21 09:42:41 1998: On ISDN to
hqLANrtr1 after 6 seconds on frame to hqLANrtr2
remote544 -> UP <- Mon Sep 21 09:43:42 1998: Up on frame
to hqLANrtr2 after on ISDN for 61 seconds
remote544 - ERR! - Mon Sep 21 09:44:03 1998: Frame up to
hqLANrtr2 21 seconds after already up
remote544 - isdn - Mon Sep 21 09:47:40 1998: Idle ISDN
call to hqLANrtr1 dropped after 0:03:58, call lasted
0:07:14

```

An example of detection of a router reboot occurs for remote 815 at 13:33:49. A little extrapolation shows that this router never reported going down, as the reboot detection reports the last syslog entry from this router as occurring at 6:38:44, while the link was reported down (by the router at the core end) at 11:41:03. Since the startup sequence is normal (ISDN comes up immediately, followed by frame relay after LMI settles down) and there are no instabilities in either function, it appears that whatever caused the router to fail and require a reboot is no longer a factor.

```

remote815 ** BOOT ** Mon Sep 21 13:33:49 1998: remote815
rebooted some time after Mon Sep 21 06:38:44 1998
remote815 -> ISDN <- Mon Sep 21 13:33:53 1998: Up on ISDN
to hqLANrtr1 after DOWN for 112.8 minutes
remote815 -> UP <- Mon Sep 21 13:35:04 1998: Up on frame
to hqLANrtr2 after on ISDN for 71 seconds
remote815 - isdn - Mon Sep 21 13:38:25 1998: Idle ISDN
call to hqLANrtr1 dropped after 0:03:21, call lasted
0:04:36

```

The client in this example tested their ISDN links by automatically placing a brief call every morning between 4:00 AM and 8:00 AM. Rather than filling the output with 150 unnecessary ISDN calls every morning, the analysis assumes that any call within that time slot is a test call if it is the first call from that site during that day's testing time slot and is under three minutes long. This allows us to keep track of test results as show below. Note that the individual lines include the date and status so that grepping for a specific site will include test results that are meaningful in isolation.

```

===== Results of ISDN Testing for Sep 28, 1998 =====
Locations Passing:
Pass Sep 28: remote011 remote012 remote013 remote014
remote015 remote016 remote018 remote020 remote023
remote046
Pass Sep 28: remotel10 remotel11 remotel12 remotel14
remotel115 remotel116 remotel117 remotel118 remotel119
remotel120
Pass Sep 28: remotel21 remotel23 remotel45 remotel46
remote211 remote212 remote213 remote214 remote215
remote216
Pass Sep 28: remote217 remote218 remote220 remote222
remote245 remote248 remote310 remote311 remote312
remote313

```

```

Pass Sep 28: remote315 remote316 remote317 remote318
remote319 remote320 remote321 remote345 remote346
remote347
Pass Sep 28: remote410 remote411 remote412 remote413
remote414 remote415 remote417 remote418 remote419
remote423
Pass Sep 28: remote444 remote445 remote446 remote447
remote510 remote511 remote512 remote513 remote514
remote515
Pass Sep 28: remote516 remote518 remote519 remote520
remote544 remote545 remote547 remote552 remote610
remote611
Pass Sep 28: remote612 remote613 remote614 remote615
remote616 remote617 remote618 remote619 remote622
remote644
Pass Sep 28: remote645 remote710 remote711 remote712
remote713 remote714 remote715 remote717 remote718
remote719
Pass Sep 28: remote720 remote746 remote810 remote811
remote812 remote813 remote814 remote815 remote816
remote818
Pass Sep 28: remote819 remote820 remote844 remote845
remote910 remote912 remote913 remote914 remote915
remote916
Pass Sep 28: remote918 remote920 remote921 remote922
remote944 remote945 remote946 remote947
Detected Failures:
Fail Sep 28: remote017 remote113 remote133 remote144
remote210 remote232 remote233 remote246 remote314
remote416
Fail Sep 28: remote420 remote421 remote422 remote433
remote453 remote522 remote620 remote646 remote716
remote722
Fail Sep 28: remote747 remote817 remote847 remote911
remote917 remote919
For a total of 128 passing and 26 failing
=====

```

The log analysis can also detect activities which should not be happening. For example, at 14:13:05 on Sept. 27, several ISDN calls were reported by remote 646 from phone numbers which were not expected to call into the site. A similar series of events occurred at 15:54:16 on Sept. 24 against remote 810.

```

remote646 * WARNING* Sun Sep 27 14:13:05 1998: ISDN call
to remote from 5165551234
remote646 * WARNING* Sun Sep 27 14:13:07 1998: ISDN call
to remote from 5165551234
remote646 * WARNING* Sun Sep 27 14:13:26 1998: ISDN call
to remote from unknown
remote646 * WARNING* Sun Sep 27 14:18:00 1998: ISDN call
to remote from unknown

```

An overall view of the state of the network is provided by the summary statistics at the end of the analysis. Failures of the T1 and PRI lines at the core are detected by keeping track of the open PVCs and calls on each interface and declaring the fault to be a T1 or PRI failure if the call count goes to zero (or near zero for frame relay, as the extreme traffic burst frequently results in lost event reports) in a very short

period of time. Following the summary statistics for the core routers are summary statistics for each remote site. The low numbers for the maximum simultaneous ISDN calls at the core indicate that there were no major events affecting a large number of remote sites.

```
=====
=====
```

Statistics for central core routers

```
s
-
-
```

Core ISDN_PRI	Router	Max up	Down	Hits	Downtime	Calls	Hi
	hqLANrtr1	44		0	--none--		
		1,128	0	6	--none--		
	hqLANrtr2	48		0	--none--		
		435	0	5	--none--		
	hqLANrtr3	59		0	--none--		
		577	0	7	--none--		

Statistics for remote locations

Location	Log	Frame drops	ISDN Test Disconnected	ISDN Statistics
	Errs? active backup		Pass Fail time total	calls idle backu
remote011	1	2	6 4	6 0:09
:34	0:02:23		0:00:05	
remote012	0	0	9 1	2 0:05
:18	--none--		--none--	
remote013	1	1	8 2	3 19:20
:33	0:03:00		--none--	
remote014	0	0	9 1	2 0:04
:58	--none--		--none--	
remote015	2	12	9 1	8 0:29
:03	0:57:44		0:00:29	
remote016	0	0	9 1	2 0:04
:14	--none--		--none--	
remote017	18	1	0 10	390 0:00
:36	6d 23:30:27		3d 00:28:42	
remote018	1	1	9 1	2 0:04
:24	0:00:38		0:00:22	
.				
.				
remote946	0	0	9 1	2 0:05
:01	--none--		--none--	
remote947	0	0	9 1	2 0:06
:53	--none--		--none--	
remote999	0	2	0 2	1 0:02
:50	0:02:57		0:00:04	

```
=====
=====
```

PROCESSING TOTALS

```
Log covers from Mon Sep 21 00:00:13 1998 to Wed S
ep 30 23:59:58 1998
  with 0 gaps of 30 minutes or longer detected.
    34,487 log file lines examined
      0 log file comments ignored
    34,487 log file entries analyzed
      0 log file lines unrecognizable
      161 locations found and included.
      99 inconsistent state changes detected.
      2 duplicates of 1 log entry detected.
    All locations had detectable frame or IS
DN activity.
      610 frame relay drops detected.
      976 authenticated, non-
test ISDN calls detected.
      2138 ISDN calls of any type detected.
      7 Security warnings issued.
      Apparent log reporting delays of at
least 61 seconds detected.
      No integrity check failures.
```

Many useful inferences can be made from the summary statistics. For example, we see an indication that remote 013 had unreported frame relay down time (based on over 19 hours of unneeded ISDN connect time), which would cause us to go back and look for a problem with the site if we had not already noticed it in the review of the detail data. We also see from the high number of ISDN calls and the time spent active on ISDN (or down) that remote 017 has no frame relay service and is having problems with its ISDN service. We also see that every location seems to have at least one ISDN test failure, which is unusual. Looking at the ISDN test results in the detail section, we discover that only one remote is listed as passing the testing on the morning of Sept 30. More likely, this location was making a brief ISDN call due to an unreported frame relay glitch and for some reason ISDN testing was not conducted that day.

Bottom Line

While a quick and dirty *perl* program can be useful to extract specific information from a specific network's log files, considerable additional effort is required to build a generally useful tool. Simple single link monitoring and extraction of "interesting" events work very well and the techniques used can be easily modified to monitor other specific needs. General analysis of the performance of an entire network is significantly more difficult, as the analysis relies heavily on the specifics of the network being analyzed and usually requires extensive code customization. On the other hand, a frequent payoff from automating syslog analysis for an entire network is the modeling of the network configuration and operation required to correctly analyze variations and event sequences found in the logs of day to day operation. Mistakes and exceptions in the network design are

frequently uncovered when debugging incorrect results from an analysis run.

Of course, care must be taken to interpret all results provided with an eye on reality as you never know when a combination of events will occur which exceeds the capabilities of any program. But analyses do not need to be perfect in order to be useful. By winnowing out much of the noise in the syslog files while retaining the useful information and presenting it in a coherent fashion, automated analysis allows the network manager to concentrate on what is happening in the network and ferret out those anomalies which deserve closer attention.

By analyzing far more data than could ever be looked at manually, automated syslog analysis allows total quality control concepts to be effectively applied to routine network operations. The hard numbers provided can be used to drive trend analyses, pinpoint chronically weak links, weather related problems, and the impact of vendor or configuration changes. Routine use has also uncovered IOS bugs, faulty test procedures, lapses in configuration management, and attempts at system access by unauthorized users.