An approach to best-in-class interoperability testing.

Nancy Griffeth
Fred Stevenson

1 Abstract

Modern networks are increasingly complex and diverse. They contain both legacy equipment and equipment based on state-of-the-art technology. They typically contain equipment from dozens of vendors. Coordination and communication between the different devices use any of a multitude of protocols. Correct interworking of technologies, protocols, and devices is problematic.

We do interoperability testing to ensure that a heterogeneous network works as required. In this paper, we explain the problems that make interoperability testing difficult and describe how to address them. We define a best-in-class process, used in our organization, which supports systematic management of the problems of interoperability testing.

2 Introduction

Testing a network entails connecting network elements and determining whether they work as required. Most of the currently available publications on network testing address how to test individual devices or technologies. For example, a good reference for testing a data network is [The Art of Network Testing]. For circuit-switched technology, see [Communications Network Test & Measurement Handbook or Test solutions for digital networks].

The history of networking in the second half of the twentieth century is a story of increasing heterogeneity, as existing networks evolved from analog voice networks to digital voice networks, from circuit-switched to packet-switched networks, from wireline to wireless networks, from electronic to optical networks. New technologies introduce heterogeneity because it is not possible to install them everywhere at once; instead, they must be introduced gradually and interwork with the pre-existing network.

There are at least two ways that network elements can fail to interoperate in today’s network. First, the implementation of similar elements from different vendors (for example, two routers from different vendors) may differ enough that the elements cannot talk to each other correctly. Second, the elements may be able to talk to each other, but they may not deliver the expected service.

The earliest voice and data networks were the product of careful design, but the paradigm for building a network today depends on integration rather than design. That is, independent companies develop useful network elements or services. An integrator recognizes the value that could be gained from putting them together, and designs a network based on pre-existing elements. However, the elements may not interoperate. This requires testing to determine whether the elements work together.

In this paper, we focus on testing service provider networks. Service provider networks provide their customers with access to a network, which could be the Internet, the Public Switched Telephone Network (PSTN), or a combination.
The rest of the paper is organized as follows:

Section 3, Background, reviews the technology and industry trends that have led to today’s heterogeneous service provider networks.

Section 4, Interoperability Testing, discusses the role and problems of interoperability testing.

Section 5, The Testing Process, describes the testing process used in Lucent’s Next Generation Networking department and outlines how the process is used to ensure that the problems of interoperability testing are addressed in a systematic way.

Section 6, Summary, draws some conclusions and raises some research issues.

3 Background

4 Interoperability Testing

4.1 Why Test Interoperability

Since service provider networks may contain network elements from many different vendors, there is a possibility that errors will be introduced by differences in implementation between different vendors or even different versions of a product from the same vendor. Also, if the network provides a new service based on independently-developed elements, the elements may not be able to talk to each other or, even if they can talk, they may not provide the desired service.

One might well ask why network elements should fail to talk to each other, when there is so much work in the standards arena to ensure that they do. The reasons for this include both business and technical reasons. We will not address the business reasons for it, but just note that businesses sometimes view features that defeat interoperation as a competitive advantage.

The technical reasons include:

1. Ambiguity in the standards. Frequently, standards provide choices to implementers, and different choices can defeat interoperation. Other, unintentional ambiguities exist in standards also.

2. Standards cannot be developed as rapidly as ideas for new services. Sometimes a new service will hit the market before any standard exists for it, and the protocols involved may become informal, de facto standards. However, these protocols can also evolve in ways that defeat interoperation.

3. While conformance to a standard for a protocol may guarantee that devices talk to each other, it doesn’t guarantee that they will interoperate to provide an intended service.

For all these reasons, even the best network design may be defeated by the implementation of some of the network elements. Thus it is crucial to perform end-to-end interoperability testing of the services to be provided by the network before deploying the network.
4.2 Objectives of Interoperability Testing

The ultimate objective of interoperability testing is to ensure that a network provides the desired services. This amounts to “acceptance testing” of the network.

However, acceptance testing is done when a customer is purchasing a network. It is necessary for a network integrator to test a planned network even before marketing it. Here are several useful types of testing that may be done in the early stages of network design:

1. **Pairwise Testing**: To determine whether specific network elements can be used in a network architecture and to determine whether the network elements communicate correctly when used in pairs.
2. **Application and Functionality Testing**: To determine whether the network provides the required services.
3. **Performance Testing**: To determine that the network performs as required.
4. **Reliability Testing**: To determine whether the reliability and availability of the network are adequate.
5. **Network Management Testing**: To determine whether the network is manageable.

**Acceptance Testing**: To verify that the network meets basic requirements of a customer. All of the above types of testing may occur as part of an acceptance test.

**Regression Testing**: To determine whether a new version of a network (or a new version of a product in the network) impairs the network.

We do all of these kinds of testing in NGN.

4.3 Problems in Interoperability Testing

A number of aspects of interoperability testing present difficult problems to the tester. Here are seven issues that we have encountered in our testing in NGN. An earlier paper [Networks 2000] based on NGN experiences also addressed a subset of these issues. Later in the paper, we will describe how we address these problems.

1) **Network complexity**: A single network may contain dozens of network elements, from several different vendors. The range of network size, in our tests, has ranged from two elements (for pairwise testing) up to 50. The number of vendors in a single test network has ranged from 1 to 6. These network elements must interoperate on every applicable network layer, using every required protocol. Normally, we are concerned with the lower three layers, plus specific applications running over a networking layer. Often, the interfaces and protocols are compatible only for certain combinations of releases, so this adds to the complexity of setting the system up. The size and complexity of the network requires both variety and depth of expertise. Configuration of the network elements to obtain the required connectivity is a difficult, error-prone, and time-consuming process.

2) **Scalability of testing**: The cost and complexity of putting together an entire network for testing suggests that pair-wise testing would be more cost-effective. But network elements tested by
themselves or even pair-wise may fail when combined in certain ways. For example, we observed that a Media Gateway Controller (MGC) that conformed to the H.248 protocol worked correctly with an IP gateway, but not with an ATM gateway. The failure probably would not have been detected without extensive end-to-end testing, because it happened only after a call to a busy line was terminated. The MGC was trying to re-use an ATM circuit, which is possible with an IP socket. The result was failure on subsequent calls.

Because of the number and variety of network elements and protocols, determining which network configurations to test and what tests to run is difficult and time-consuming. The number of tests that can reasonably be used is staggering. Just to determine that a single network element conforms to a single protocol can require thousands of test cases. When trying to establish that an entire network runs correctly, some means of deciding which tests will provide adequate information about system behavior is required.

3) **Inadequate requirements**: We have found that requirements for a network solution are sometimes inadequate to develop a test plan. Sometimes, it is possible to use standards or well-known behaviors of existing systems in place of requirements (for example, when testing a network providing Voice over IP, the behavior of voice systems is standardized and well-known). However, network testing can sometimes be prototyping, especially when cutting-edge technologies are involved: The goal in these cases is to determine what the network does, not to determine whether it meets certain requirements.

4) **Determining root causes of defects**: When the network fails a test, the determination of which network element is at fault is often controversial. To complicate matters more, it is possible that the difficulty is in the interaction of the boxes, and that some compromise is necessary to determine how to change the network to pass the test.

Also, for quality purposes, someone should determine the root cause of the failure. It is problematic to get the required information, when much of the process of developing the network elements is proprietary to different vendors. It is also problematic to determine who should do the root cause analysis.

5) **Accurate Measurement**: Cost-effective measurement of certain properties of a network may challenge the state of the art. For example, how does one measure the availability a network that is allowed at most 15 minutes of down-time in a year?

Voice quality on packet networks is another area where measurement is difficult. The ITU provides two standards for this, one for subjective and one for objective testing. The subjective test is time-consuming and expensive, but provides a good handle on how actual customers will perceive the voice quality. The objective testing is automated and (ignoring the capital expense of the test equipment) it is quick and cheap. However, the relationship of the number provided by the objective tests to customer perception of voice quality is unknown [reference to Pennock].

6) **Testing the Test Equipment**: How can we be sure that a test passes or fails, unless we are sure that the test equipment is working correctly? There are actually several issues embedded in this: First, the test equipment must do what it was built to do. Second, the measurement that it was built to take must reflect the system properties accurately (in the case of
objective measures of voice quality, this remains to be established. Third, for simulators (including load generators), the model of the environment must be accurate. This last is problematic when using commercial equipment to generate load for packet networks, where traffic patterns are very complex [references to Leland et al, Cleveland].

7) **Documenting Test Results and Learnings:**

In a major test project, we may run anywhere from several hundred to several thousand test cases. Many of the test cases may consist of multiple sub-tests, each of which must run successfully for the overall test to pass. Because of the sheer numbers and complexity of the tests, it is impossible to capture test results without careful documentation. Also, specialized monitoring equipment may be necessary to capture exactly what happened in some parts of the network. For some projects, we also provide implementation and release notes. These also cannot be created without careful documentation.

These are all difficult issues, and we do not believe that they can be totally solved. However, we can ameliorate the difficulty with appropriate processes, methodologies, and tools. In the next section, we discuss processes.

## 5 The Testing Process

### 5.25.1 Process Flow

In NGN, we have developed a defined sequence of process steps to be used across all projects. The first two process steps in the diagram are strictly administrative, so we ignore them here. The subsequent process steps are:

1. **Launch Project**: Formalize a Statement of Work, get Billing Agreement signed, and set up project management infrastructure.
2. **Setup Test Lab**: Order equipment and install and configure the network in the test lab.
3. **Develop Test Plan**: Write the test plan and define the test cases and procedures.
4. **Execute and Monitor Tests**: Execute the test plan.
5. **Document Results**: Write up the test results and other deliverables.
6. **Evaluation**: Complete project management database, free up resources, and evaluate performance on the project.

The diagram in Figure 1 illustrates the process flow.
Because we have a defined process, we can determine where we can best address each of the issues raised above. We can also collect metrics to evaluate our effectiveness in testing. The metrics we collect are also used for estimating project time and resources and for process improvement.

Table 1 summarizes how the problems raised by interoperability testing can be addressed within the NGN process.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Process Step</th>
<th>Process Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Complexity</td>
<td>1-Launch Project</td>
<td>Negotiate a realistic network architecture or architectures for testing</td>
</tr>
<tr>
<td></td>
<td>2-Set up Test Lab</td>
<td>Make sure that appropriate skills are available for required setup activities Use network administration tools to automate setup Record configurations from successful tests to use as reference and also to re-use in subsequent tests Record configurations from unsuccessful tests for reference and analysis</td>
</tr>
<tr>
<td></td>
<td>3-Develop Test Plan</td>
<td>Use tools for generating a comprehensive collection test cases</td>
</tr>
<tr>
<td>Scalability</td>
<td>2-Setup Test Lab</td>
<td>Maintain a database of configuration files</td>
</tr>
<tr>
<td></td>
<td>3-Develop Test Plan</td>
<td>Use tools for minimizing test cases</td>
</tr>
<tr>
<td></td>
<td>4-Execute Test Cases</td>
<td>Automate test execution</td>
</tr>
<tr>
<td>Table 1. Addressing problems in interoperability testing within the defined process.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Inadequate Requirements</strong></td>
<td><strong>Agree on source of requirements or negotiate agreement that the work is prototyping rather than testing</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Analyzing failures</strong></td>
<td><strong>Send MR’s to product organizations when failures are discovered</strong>&lt;br&gt;<strong>Work with product organizations to analyze root causes of problems found</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Accurate Measurement</strong></td>
<td><strong>Study available tools and techniques</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Testing the test equipment</strong></td>
<td><strong>Determine how field defects could have been found in testing</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Documenting Test Results and Learnings</strong></td>
<td><strong>Setup necessary monitoring equipment, auxiliary PC’s for maintaining test logs, and LAN connections to testers’ desktops</strong></td>
<td></td>
</tr>
<tr>
<td><strong>6-End Project</strong></td>
<td><strong>Record and analyze test results from specific tools</strong></td>
<td></td>
</tr>
<tr>
<td><strong>3-Set up Test Lab</strong></td>
<td><strong>Keep test logs using lab infrastructure facilities</strong></td>
<td></td>
</tr>
<tr>
<td><strong>4-Execute Test Cases</strong></td>
<td><strong>Use test logs to create implementation notes and engineering guidelines</strong></td>
<td></td>
</tr>
</tbody>
</table>

The problem of network complexity has a side-effect on the test process that we did not expect, so we discuss it here. The chart in Figure 2 shows the time spent in each of steps 4-7 (where the project is actually executed). We would have predicted that test planning and execution would take most of the time, but in three of the four projects, lab setup (configuration and provisioning) took more than half the time. In only one project was this not the case. However, that one project was itself a test of configuration and provisioning of the converged network under test.

Configuration and provisioning are thus important issues for interoperability testing. They must be addressed in order to improve the quality and reduce the testing interval.

The approaches that we recommend are:

**Automation:** Use network administration tools to automate as much as possible of the configuration process.

**Documentation:** Record configurations used in successful tests. Also, record configurations that did not work and why. The latter information can be used to create release and implementation notes.

**Improved network management:** This is beyond the scope of a testing organization, but when network management systems have been improved this part of the testing process will be shorter and less error-prone.
Each phase of testing has special problems; solving these problems can be simplified by use of appropriate tools.

**Test planning: Test case generation tools:**

Test planning requires determining which tests to run. Providing adequate coverage is a major issue. Getting the tests done on time is also a major issue. Tools for generating a comprehensive set of test cases satisfying given criteria can address both of these issues. A useful technique for generating a minimal set of test cases covering all possible pairs of combinations is called “orthogonal arrays” (Phadke, Dalal). As input, the tester prepares a list of all possible values of each factor (e.g., protocol, interface, or network component) in the test. The output is a list of combinations to try, which is guaranteed to cover all combinations of values for any two selected values.

A second technique to ensure coverage of all functions of an end-to-end application is to model the application as a state machine, and use a tool called ITIS (Griffeth, Hao, Lee, and Sekar) to generate a minimum number of test cases to cover the functions. The reduction in test cases is significant.

ITIS assumes that system testing has uncovered all single-system errors, and uses only those cases that exercise interoperation between network elements. For Voice over
Packet testing, we used a finite state machine model of voice telephony and reduced the number of test cases from 1752 to 22.

**Test execution: Load modeling**

Performance, reliability, and availability testing all require modeling a realistic system load and generating it. There are significant obstacles to accomplishing either of these tasks with adequate quality.

For testing networks that support new applications, load modeling is guesswork. But for testing networks that support popular applications such as email or Web browsing, the load model can use information derived from operational networks. This is non-trivial, but data is available and there are techniques for analyzing it.

A number of research groups have done studies of Internet traffic. One of the most frequently-cited papers on the characteristics of Internet traffic is [Leland et al], showing that Internet traffic is self-similar, i.e., chaotic. This presents some challenges to analyzing and modeling Internet traffic. However, a more recent study from The Bell Labs Internet Traffic Research Group shows that Internet traffic can be accurately characterized by traditional statistical models [Cleveland et al].

Commercial load generators provide a simple means of generating load, but they are expensive and the load models are simplistic. Commercial vendors should be encouraged to take advantage of the work cited above in characterizing Internet traffic. Alternatively, custom load generators can support a more accurate load model, but are time-consuming and expensive to develop and support.

**Test execution: automation**

To determine whether a test case should be automated, consider the following factors: *How often is the test used? Will it be used regularly, or is it a one-time test case?* The start-up cost of automation is high, so that tests should not be automated unless they will be used fairly frequently.

*How long does the test take to execute? What percent of the test time in the organization does the test take?* If the test case doesn’t take a significant percentage of testing time, attention would be better directed elsewhere.

*How error-prone is the procedure required to perform the tests?* If a test is very difficult to run manually, then automation should be considered.

*Can measurements be taken accurately if the test is done manually?* Some tests require very precise measurements of timing, and so the steps of the test must be automated to record the time accurately.

**Test execution: Monitoring and analysis tools**

Many important characteristics of a network cannot be determined without the use of specialized monitoring tools (for example, bandwidth utilized, cell or frame loss, and delay). Other characteristics, such as voice quality, are time-consuming and expensive to measure in the traditional way. Unfortunately, as discussed above, the new methods for measuring voice quality are not yet proven out.
5.15.3 Measuring Process Effectiveness

An effective organization evaluates its processes to ensure that it continues to be effective. For an evaluation to be meaningful, the evaluation must use metrics that measure important aspects of the process, that are well-defined, and that can be compared across different organizations and projects. We discuss some such metrics here.

5.1.45.3.1 Defect-Related metrics

The purpose of the testing process is to uncover defects in the network. The most direct measures of effectiveness will address defects found and defects not found. We look at three kinds of metrics in this category:

1) Defects found
2) Root cause of defects found
3) Defects not found (in testing)

Analysis of these metrics supports improving the development process by isolating where defects are introduced for process improvement; determining when to stop testing (when expectation of finding more defects is low); and evaluating the effectiveness of testing in reducing defects found in the field.

Keeping track of defects found and their root causes supports various sorts of analysis.

---

Figure 3. Introduction and Removal of Defects in the Network Life Cycle
We discuss analysis of the defects found for the following two purposes:

1. Determining the effectiveness of the test organization in finding and removing defects.
2. Deciding when to stop testing.

5.3.1.1.1 Determining the effectiveness of the test organization

One tool for evaluating the effectiveness of the process of integrating a network is the “life-cycle root cause” graph, illustrated in Figure 3.

Each box in the graph represents a phase of the network integration life cycle. The arrows above the boxes represent defects added to the network at that stage. The arrows at the bottom represent defects removed at the stage. The arrows pointing from one box to the next represent defects passed on to the next stage. For our purposes, we can measure the effectiveness of the integration test lab by looking at defects added, defects removed, and residual defects.

Using this model requires cooperation of many organizations. In order to assign a defect to the stage in which it was introduced, a root cause analysis must be done for each defect. This requires cooperation of product development organizations and requirements organizations. Also, determining residual defects requires cooperation from field support.

5.3.1.1.2 Deciding when to stop testing

In software and system testing, a common practice is to graph defects found versus time. A variant is to chart defects found versus test cases run. Testing organizations use these curves to determine when to stop testing. An S-curve rises rapidly for a while, but then it flattens. When it flattens, further testing is not cost-effective. The left-hand side of Figure 4 illustrates this idea.

However, this works best when the number of defects is very large. By the time we do network integration testing, the individual software packages and network components have been thoroughly tested; we find only the errors caused by incorrect interworking. The number of defects that we typically find in network integration testing is normally so low that it’s statistically impossible to determine whether the curve is flattening. The curves we actually see look more like the right-hand side of Figure 4.

---

**Figure 4.** Theoretical (left) and actual (right) S curves
For this reason, determination of when to stop testing is quite difficult for network integration testing. We suggest developing a method for evaluating coverage (see Section 0 for more discussion of this).

### 5.1.25.3.2 Resource-Related metrics

Resource-related metrics should be collected at least for:

1. Total project time and time in each phase of the project (see Figure 1).
2. Time using network equipment (this should include all time that the equipment is unavailable to other projects, i.e., it should include setup time) and cost of network equipment, in each phase.
3. Software used and purchased, in each phase.
4. Staff time, in each phase.
5. Consulting time and costs, in each phase.
6. Test tool use (as above, total time unavailable to other projects), in each phase.

These metrics support estimating project time, staff, and cost for future projects. They also support improving the testing process by focusing attention on the most time-consuming or most expensive parts of the process. Figure 5 is taken from a study of the NGN process. In this study, we focused on the four stages that we know from experience take the bulk of the time. Projects 1-4 show the times in four actual projects. The bars above 5 give the average of the four projects.

We were surprised that test setup took as much time as it did. This finding has directed us to concentrate on improvement on this area.
Figure 5. Breakdown of projects by time in each project stage

5.3.3 Test-Related metrics

Test-related metrics can help to evaluate the theoretical effectiveness of the testing and to improve the test process by understanding what kinds of tests need automation to reduce their time or improve their quality. The metrics are:

1. Number of test cases executed, including the test result
2. Time per test case
3. Test coverage

5.3.3.1 Test process improvement

Experience is the best teacher, and testing experience should be no exception. Tracking information about test cases and their disposition can reveal several important pieces of information:

**Which test cases are most likely to fail.** Providing this information to product development units can help them to develop a better product. Also, using this information in the testing organization can improve the quality of the testing. This is because tests that have failed frequently are likely to exercise network components and subnetworks that are difficult to integrate. Thus tests that have failed in the past should be emphasized in subsequent testing, while tests that rarely fail may be condensed and in some cases even skipped. For example, a tester could safely skip a network connectivity test that almost always passes, since the failure will be discovered in subsequent tests anyway. On the other hand, if it fails frequently, the tester would want to execute it independently to simplify analysis of the failure.
Which tests take the most time. For example, manually provisioning a circuit in a network takes much longer than simply placing a voice call. Techniques for simplifying and automating the most time-consuming tasks must be considered. Also, using tools to generate test cases or to automate testing should be considered.

Which test procedures are most difficult to execute correctly. Testers may find it difficult to perform certain test procedures correctly. If this is the case, test automation may be an alternative.

5.3.3.2 Effectiveness of testing

In Section 5.3.1.1.1, we pointed out that S curves are not very helpful in integration testing, because almost all of the software and single-system defects have already been removed. Only the interworking defects remain, and these are usually relatively few in number in a given network (but can still be catastrophic in their effects).

We propose an alternative approach to evaluating the effectiveness of the testing, which is to evaluate the theoretical coverage. To do this, we need to find a way to extend the idea of test coverage beyond the way it is used for software, where various kinds of coverage are well-known (such as all-statements or all-paths). When we talk about determining the coverage of a set of test cases applied to an integrated network, we are looking at:

The interfaces (e.g., DS1, DS3, FastEthernet, GigabitEthernet)

The protocols (e.g., SS7, IP, SIP, TCAP, MPLS, H.248)

Component variants (e.g., POTS phone versus IP phone; network elements from different vendors)

The execution paths of an application (e.g., voice telephony, Web browsing)

In a typical network, there may be hundreds of possible combinations of interfaces, protocols, and network components. For example, in the voice over packet architecture illustrated in Figure 6, there are a number of different SS7 variants that could be used; ATM (ABR, CBR, or VBR) or IP (IPv4 or IPv6) could be used for the packet network; the voice over packet signaling protocol could be H.323, H.248, or SIP; there are various choices for the layer 2 network. There are many different voice gateways, circuit switches, and softswitches that could be used in this configuration.

How can we verify that we have tested all of the important combinations, and do the testing in a reasonably efficient way? See Section 5.2 for more information.
6 Summary

A best-in-class process requires understanding the goals of the process, establishing a well-defined sequence of steps that are published and enforced, using metrics to measure the outcome of the process.

For interoperability testing, a best-in-class process requires that we address the problems of complexity, scalability, inadequate requirements, hard-to-find failure causes, measurement difficulty, unreliable or non-existent test equipment, and extensive documentation needs.

We have recommended approaches to address most of these problems, including time and thought at certain points in the process flow; the use of appropriate tools; and the use of metrics to discover the most important places to address the problems.

Some of the problems mentioned here are still not particularly tractable, notably:

1. Determining voice quality
2. Generating custom loads
3. Configuration of the network

Research will be required to improve testing in these areas.

Acknowledgements. We would like to acknowledge Phil Stern for his work in developing the NGN Process Flow and for his comments and advice on this paper.