DYSWIS: An Architecture for Automated Diagnosis of Networks[1]

Vishal K. Singh, Henning Schulzrinne
Dept. of Computer Science, Columbia University
{vs2140, hgs}@columbia.edu
Kai Miao
Intel Corporation, Santa Clara
Kai.miao@intel.com

Abstract

As the complexity of networked systems increases, we need mechanisms to automatically detect failures in the network and diagnose the cause of such failures. To realize true self-healing networks, we also need mechanisms to fix these failures and ensure service availability by providing alternatives when a failure is detected. In this paper, we address the detection and diagnosis of network failures. We introduce DYSWIS ("Do you see what I see") which observes the system from multiple points in the network. Our system consists of detection nodes and diagnosis nodes. Detection nodes detect failures by passive traffic monitoring and active probing. Diagnosis nodes determine the cause of failures using historical information about similar failures and by performing active tests. They are based on a rule engine and represent network dependency relationship encoded as rules. We present our prototype system which considers network components present in a VoIP network and show the feasibility of our solution.

1. Introduction

Failure diagnosis is one of the major challenges that home users and network administrators face today. Many different components which collaborate to realize a particular service and these components belong to different administrative domains as well as physical locations. With increasing number of such services, it is important to design systems which enable easy diagnosis of problems encountered and allow determining the root cause of the failures.

To diagnose network problems, this paper proposes a system called ***DYSWIS ("Do you see what I see")***. DYSWIS leverages distributed resources in the network. It treats each node as a potential source of network management information, gathering data about network functionality. The state of the network is observed by topologically dispersed nodes in the network. Multiple views of different parts of the network are aggregated to get an overall view of the network. Once a diagnosis node has gathered insights on whether other systems are experiencing similar problems, it then combines this information with local knowledge and tries to estimate root causes, using a rule based system, rules represent the dependencies among various network components.

DYSWIS nodes differ in capabilities from basic failure detection node to maintenance of failure history records to the ability to invoke a set of standardized or customized network probing tools within the system for specific network and application layer protocols and the ability to learn and track network fault behavior, create and manage diagnostic tests based on dependencies between network components or protocols. A subset of DYSWIS nodes in a network possess analytical capability allowing them to make inferences about a particular failure condition, perform or request diagnostic tests towards localizing a specific type of fault or faults and then draw conclusions regarding the nature and scope of a particular fault.

The remainder of this paper is organized as follows: Section 2 presents related work; Section 3 presents the DYSWIS approach, Section 4 presents a diagnosis flow example, Section 5 presents implementation details, followed by future work in Section 6 and conclusion in Section 7.

2. Related Work

Current fault diagnosis systems derive information using SNMP [6] and other tools like ping, traceroute, and dig. However, these tools provide no assistance to end users or network administrators in identifying the source of the problem in a multi-provider systems leading to rather pointless ``try this'', "re-install" or "reboot that" exercises that frustrate users and cause significant costs in technical support. Other issues with current fault diagnosis systems are that they are manual, time consuming and centralized. Centralized management infrastructures are vulnerable to single point of failure and have a single view of the entity being managed. AutoMON [1] uses a P2P-based framework for distributed network management. It

[1] This work was supported by Intel Corporation.

uses distributed network agents to test the performance and reliability of network. The nodes are not intelligent, do not co-operate or use historical information about failures. Irina et. al., [2], proposes an architecture for real-time problem determination using active probing and proposes a probe selection criteria and algorithm. Beygelzimer et. al. [3] propose algorithms for representation of knowledge about the system. The dependency relationship among network components and services is one such kind of knowledge. Chen et. al., [4], proposes to use decision tree learning for failure diagnosis. Distributed fault management in Connected Home [5] systems does fault diagnosis for connected home system using an agent-based approach. It uses a set of tests which test the different internal components and are pre-defined like rules for each subsystem.

3. DYSWIS Approach

We modeled the fault detection mechanism on the medical diagnosis system and the cooperation between peer nodes is modeled on human social network system. A medical diagnosis of a patient by a doctor involves analysis of symptoms, certain diagnostic tests to isolate or confirm possible causes, in addition to leveraging knowledge already available in the world of medicine and knowledge from tests already done in the process of diagnosis. Similarly, to find the root cause of a service failure in multimedia services, it requires an understanding of the network and network components and dynamic relationships among the components, the right tools and methodology to find the root cause from the known or observed failures and from past historical information about the network.

To give an analogy of DYSWIS system with human social network system, let us consider that a user cannot connect to a web site; he naturally asks his colleagues if they are also experiencing similar problems. He would run some basic tests like a ping test from his PC to the web server's address. If the ping-based reach-ability test succeeds, he may ask his colleague whether he is able to browse the site. This will help him infer if the problem is specific to him or others are also encountering the problem. If the colleague can browse the site, the user deduces that problem is local on browser or the web site has blocked his IP address. Similarly, in DYSWIS system, a node encountering a failure first asks a neighboring node if they are also experiencing similar failures. The peer nodes, based on their past experience with the same service or based on a probe, conclude that that failure is local to the node. The diagnosis infrastructure may request multiple peer nodes about a particular service to localize the problem. Thus, DYSWIS nodes [11] try to determine the cause of failure by asking questions to peer nodes and performing active tests. The questions or queries encapsulate the dependency relationship. ***This iterative process of isolating cause of failure based on asking questions is the basis of our approach.***

3.1 Steps in DYSWIS Diagnosis

The architecture of the proposed fault diagnosis framework consists of following components: Detection and reporting of failures, pre-diagnostic processing, diagnostic test selection, executing tests, result analysis and storing historical results. Figure 1 shows a flow diagram of the diagnostic process. The fault diagnosis framework reports user detected and automatically detected failures for analysis to the fault diagnosis system. The failure reports include detailed context information about the failure, such as, one hop distance at different OSI layers, e.g., access point or switch information at layer 2, default router or subnet information at layer 3, first hop SIP proxy server at application layer, the timestamp when failure is observed, the participating node's hostname and IP addresses.

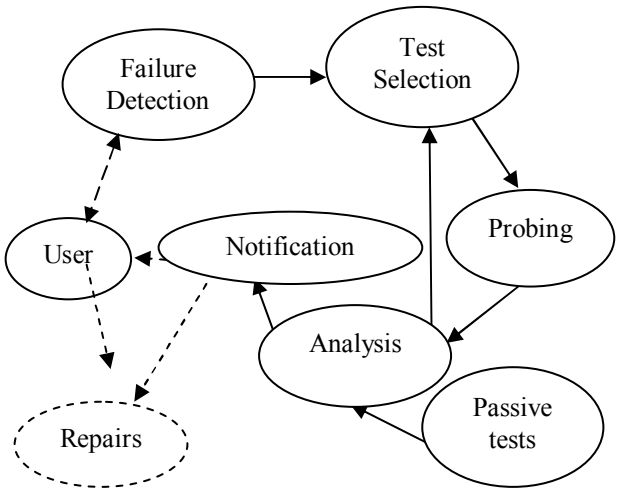

Figure 1 Flow diagram of diagnostic process

The next step is processing and storage of failure information. Failure event correlation aggregates multiple common failures as well as failures manifested by the same root cause. The received failure information is compared against existing failure information to remove duplicate diagnosis requests. This pre-diagnostic processing is used to determine what services need to be tested and results in diagnostic test selection or "Probe selection". The next step is execution of selected diagnostic test or "probing". The result of the test is sent back to the system for further analysis, which may result in selecting another diagnostic test based on the dependency graph. The results of the diagnostic tests,

both successes and failures are stored as history information.

4. Example Diagnosis Flow

Consider a user alice@example.com who tries to make a call to another user bob@destination.com and the call does not go through. The end point which tried to make a call to the remote end point asks the DYSWIS system to diagnose the problem. Following are the steps: the diagnosis node queries if any other node from caller's location has made a call to the user bob@destination.com. The response can be that other nodes have recently made a call to the destination address or to the destination domain or no nodes have made any call to destination address/domain. Based on the response, the diagnosis node may request probe node to send a SIP OPTION message or to make a call to the address. This test is done from multiple locations i.e., location of original node which encountered a call failure and other location to see if the problem is specific to the caller's location. Based on the response from the tests and historical information, the location of failure is isolated.

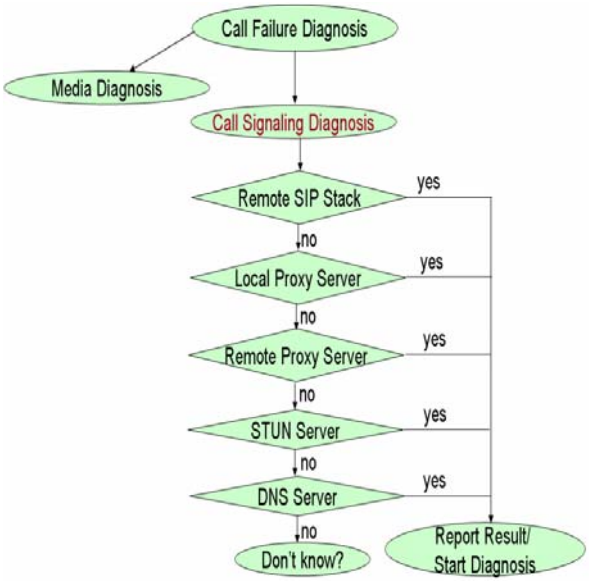

Figure 2 Call failure diagnosis

Figures 2 and 3 explain the above described approach. The flow charts show the order of tests based on a dependency graph. Figure 2 shows the rules for 'call failure diagnosis' which in turn may trigger 'media failure diagnosis' (Figure 3). The call failure diagnosis involves testing of remote SIP end point, proxy servers, STUN and DNS servers. These may further trigger diagnosis of network connectivity and supporting services.

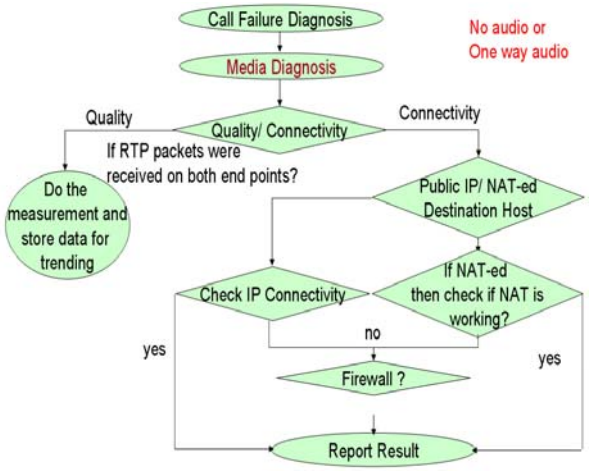

Figure 3 Media failure diagnosis

5. DYSWIS Implementation

In this section, we describe our initial implementation of DYSWIS diagnosis framework and the various components which comprise a DYSWIS system.

5.1 Diagnosis Framework

The whole DYSWIS framework consists of diagnosis nodes, sensor nodes (probe and failure detection), rules in diagnosis node, request-response protocol between the nodes and storage of historical information. These are described next.

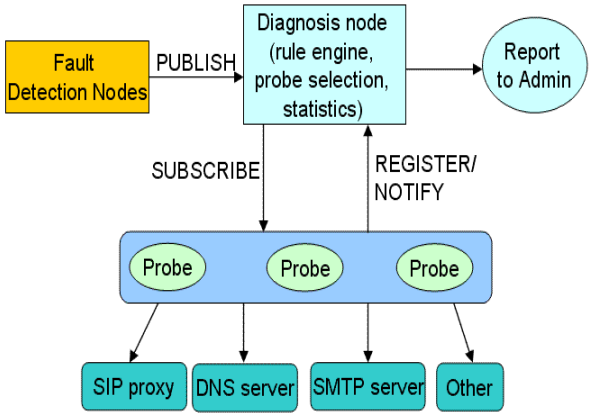

Figure 4 Diagnosis framework architecture

Figure 4 describes the flow of DYSWIS framework. Probe nodes register their location as well as testing capability with the diagnosis node. Diagnosis nodes are the smart nodes and have information about probe nodes, store historical failure information and store dependency rules encoded as tests and queries. These are described next.

Sensor nodes detect network failures; these are active and passive nodes distributed in the network. They passively sniff traffic and pro-actively report failures. They detect non-availability of service by conducting active tests which emulate user actions e.g., making a

call. We implemented the following probes for a VoIP system: SIP trace route using SIP OPTION, SIP call test, NAT test for one way audio failure detection, ping test, RTP tests for jitter and loss to measure effect on voice quality, DNS tests (A, AAAA, SRV, MX type).

Diagnosis nodes are smart nodes in DYSWIS system and work based on a rule engine. We used the DROOL [7] rule engine which is a forward chaining engine. The logic to choose the probe nodes is implemented in diagnosis nodes. In addition rules to determine the tests are specified to them.

5.1 Request-Response Protocol

The DYSWIS nodes use a request-response protocol to communicate among themselves, for conveying about detected failures to a diagnosis node and for requesting a probe to be run and results of the probe to be sent. SIP PUBLISH is used to report failures and SIP NOTIFY is used for test results. SIP SUBSCRIBE is used to request active sensor nodes to perform the tests. The probe and diagnosis nodes register with the DYSWIS system using SIP REGISTER request. They register their capability and location which is used to determine which probes will be run on which nodes. We used NIST SIP [9] for implementing the request-response protocol for DYSWIS nodes.

5.2 Rule Engine – Dependency relationship

We represent dependency using rules, using tests and queries in DROOL rule language, an example is given below:

```
Rule "call failed"
  When
       Tm: FailureObject(failureType==CALL)
  Then
    CallFailureObject cfo = (CallFailureObject) Tm;
    Boolean r = Cfo.doPreprocessing( );
    If (r == true)
       Cfo.reportResult()
    Else {
       Cfo.setTestType(TestObject.SIPPROXY)
       Modify (Tm) ; }
End
```

The rule says when an event 'Call Failed' occurs, test SIP proxy server. The "doPreprocessing" step queries for past failures to see diagnosis is not done again for a failure of same type. Thus, queries are used to find information about past failures.

6 Future Work

There is a need to generate the dependency relation automatically; statistical mechanisms and temporal correlation of failure information can be used. Secondly, we need applications to detect and report failures in order to trigger diagnosis. Additionally, failure detection can be done using traffic analysis. Finally, failure event correlation based on dependency graph i.e., using the rules is another area of future work.

7 Conclusion

In this paper, we proposed the DYSWIS system to automatically diagnose network failures and determine the root cause of failures and presented a reference implementation for a VoIP system. We used the DROOL rule framework to represent the dependency information. The dependencies were represented as using probes and queries. Our framework uses SIP event notification framework for sending requests and receiving responses. The initial results were obtained by inducing failures manually and observing how DYSWIS triggers diagnostic processing.

8 References

[1] Binzenhöfer, A., Tutschku, K,. Graben, B., Fiedler, M., Arlos, P., "A P2P-Based Framework for Distributed Network Management", New Trends in Network Architectures and Services, LNCS, Loveno di Menaggio, Como, Italy, 2006.

[2] Rish, I. Brodie, M. Odintsova, N. Sheng Ma Grabarnik, G., "Real-time problem determination in distributed systems using active probing",NOMS 2004.

[3] Beygelzimer A., Brodie M., Ma S., Rish I., "Test-based Diagnosis: Tree and Matrix Representations", in Proceedings of IM 2005.

[4] Chen, M. Zheng, A.X. Lloyd, J. Jordan, M.I. Brewer, E., "Failure diagnosis using decision trees", International Conference on Autonomic Computing, 2004. May 2004.

[5] Utton, P.; Scharf, E., "A fault diagnosis system for the connected home", Communications Magazine, IEEE, Volume 42, Issue 11, Nov. 2004, 128 - 134.

[6] Case, J., Fedor, M., Schoffstall, M., Davin, J., "Simple Network Management Protocol", RFC 1157, May 1990.

[7] DROOL - http://labs.jboss.com/drools/

[8] Jess - http://herzberg.ca.sandia.gov/jess/

[9] NIST SIP - http://snad.ncsl.nist.gov/proj/iptel/

[10] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[11] Miao, K., Schulzrinne, H., Singh, V., Deng, Q., "Distributed Self Fault-Diagnosis for SIP Multimedia Applications", To appear in MMNS'2007, 10th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services.