DYSWIS: An Architecture for Automated Diagnosis of Networks

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 alternative means when a failure is detected. In this paper, we address detection and diagnosis of network failures. We introduce DYSWIS ("Do you see what I see") which relies on observing 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. The problem is more so because there are so many different components which collaborate to realize a particular service and these components belong to different functional 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. Each node has its own view of the network. Multiple views of different parts of the network are aggregated to get an overall view of the network. Failures seen by different nodes in the network are correlated, along with historical failure information. 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. This is done using a rule based system, where rules represent the dependencies among various network components.

DYSWIS nodes differ in capability level from basic failure detection and maintenance of failure history records to the ability to invoke a set of standardized or customized network probing tools within the system (e.g., ranging from versions of ping and traceroute to more application-specific tools) 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 certain conclusions regarding the nature and scope of a particular fault. We use the DROOLS [11] rule engine in our system to encode the dependency relationship in the form of probes or tests and queries. There can be lots of variables in a complex network diagnosis, we found it quite challenging to simplify the dependency rules as well as decouple the actual diagnosis functionality (probes (active tests) and queries (historical failure information), their implementation. One of our main goals was clear separation of diagnosis logic (rules) from the processing part.

The remainder of this paper is organized as follows: Section 2 presents related work; Section 3 presents the

DYSWIS approach in detail, Section 4 presents a diagnosis flow example, Section 5 presents implementation details, types of DYSWIS nodes, protocol they follow among themselves and rule engine details, Section 6 describes the probe selection criteria, Section 7 presents the security consideration followed by future work in Section 8 and conclusion in Section 9.

2. Related Work

Current fault diagnosis systems derive information using SNMP [7] and other tools like ping [8], traceroute [9], dig [10]. 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. Centralization of management infrastructure has two main problems. It makes management vulnerable to single point of failure and secondly, it considers a single view of the entity being managed. Also, it requires a network administrator to monitor the reported failures. AutoMON [1] uses a P2P-based framework for distributed network management. It uses distributed network agents to test the performance and reliability of network. It addresses the issues with centralized network management infrastructure. However, the nodes are not intelligent, do not co-operate or use historical information about failures. The system conducts periodic tests but does not encode dependency information.

Irina et. al., [2], proposes an architecture for real-time problem determination using active probing and proposes a probe selection criteria and algorithm. It integrates in our architecture as in our approach, DYSWIS nodes select probes based on past results and knowledge about dependency relationship. 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. In our system, the knowledge about relationship among components and services is represented as rules in the form of tests and queries. 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

Before we go into describing our approach for diagnosis, we classify the causes of failures seen in the network. Some of the main causes of problems in the networks are configuration problems, software bugs, transient problems and hardware faults. The main focus of our approach is to diagnose 'transient problems' as these are most difficult to diagnose. Moreover, software bugs and configuration problems are repetitive in nature and disappear once identified and resolved. Transient problems are run time problems which go unnoticed and contribute to overall level of reliability or level of uptime and hence, customer satisfaction.

3.1 Underlying Model for 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 where the patient experiences certain symptoms of an illness, but the cause of these symptoms must be identified by a trained doctor through a methodology which may involve certain diagnostic tests to isolate or confirm possible causes, in addition to leveraging knowledge (similar medical cases) already available in the world of medicine and knowledge from tests already done in the process of diagnosis (medical history). Similarly, to find the root cause of a service failure in multimedia services, it requires an understanding of the network and network components that embody these services and dynamic relationships among the networking components and having the right tools and methodology to find the root cause from the known or observed symptoms (failures). It also involves leveraging knowledge about other failures in the network (past failures) and historical information obtained by conducting diagnostic tests.

To give an analogy of DYSWIS system with human social network system, let us consider that a user encounters a problem, that, he cannot connect to a web site; he naturally asks his colleagues or other neighboring users 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 sitting next to him, whether he is able to browse the site. This will help him to 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 or a possible firewall problem. Similarly, in DYSWIS system, a node encountering a failure first asks it neighboring node if they are also experiencing similar failures before reporting the problem for diagnosis.

DYSWIS nodes 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***.

3.2 Steps in DYSWIS Diagnosis

The DYSWIS approach relies on the peer nodes to determine the root cause of the failure. Upon encountering a failure a node asks its peer nodes if they are also observing the failure. 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 DYSWIS nodes are described in [18]. In some cases, the failure can be local to a subnet, access switch, access point or the domain. In other words, locality of failure can extend from node itself to the entire domain. The diagnosis infrastructure may request multiple peer nodes about a particular service to localize the problem.

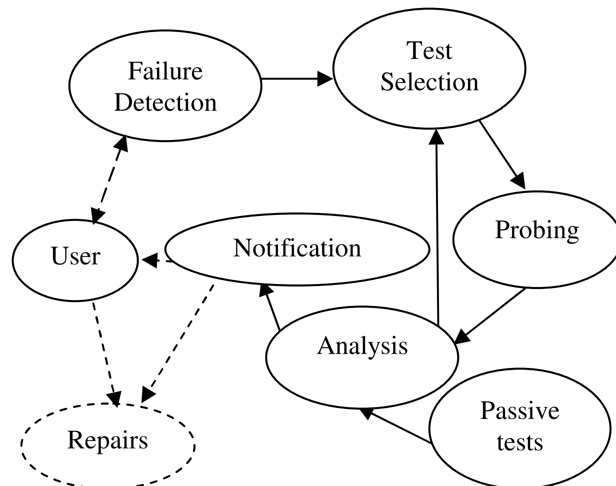

Figure 1 Flow diagram of diagnostic process

The architecture of the proposed fault diagnosis framework consists of the following major functional components: Detection infrastructure and reporting of failures, pre-diagnostic processing and diagnostic test selection and finally diagnostic tests, result analysis and storing historical results.

Figure 1 shows a flow diagram of diagnostic process. The first step in diagnostic process is detection and reporting of failures. The fault diagnosis framework reports user detected and automatically (programmatically) 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, timestamp when failure is observed, participating node's hostname and IP addresses.

The next step is processing and storage of failure information. "Failure event correlation" [17] is done to aggregate multiple common failures. The received failure information is compared against existing failure information sent from other sources to remove duplicated diagnostics for the same root cause. For example, two different phones can experience call failure because of DNS failure. Based on this pre-diagnostic processing, it is determined as what services need to be tested. Factors considered for pre-diagnostic processing can include time of failure, locality of failure, historical data about last seen failure and historical data about latest successful operation.

Once the pre-diagnostic processing is done, the next step is diagnostic test selection or "Probe selection". This determines which probe/test to run on which nodes.

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 dependency graph. If the diagnostic test results in finding the root cause of failure, notification is sent so that repair action can be taken. The results of the diagnostic tests, both successes and failures are stored as history information. The system relies on diagnostic tests and historical data of other peers. The diagnostic tests can be carried out by the same node or different nodes depending upon node's physical location, capability to perform a test and other factors. For example, a node running on a PC can perform a test to check the availability of a DNS server but may not be able to perform SIP proxy server availability testing.

4. Example Diagnosis Flow

To explain how our DYSWIS system works, consider a VoIP system (Figure 4). A failure seen by a user, e.g., a

call set up failure, can be because of access network failure, mis-behaving NAT or failure on SIP proxy server or STUN server authentication failure. Each of these failures could be caused by mis-configuration, software bugs, server or network overloading or other transient problems in the network. Additionally, there is a complete set of supporting services in the network such as DHCP, ARP, and DNS.

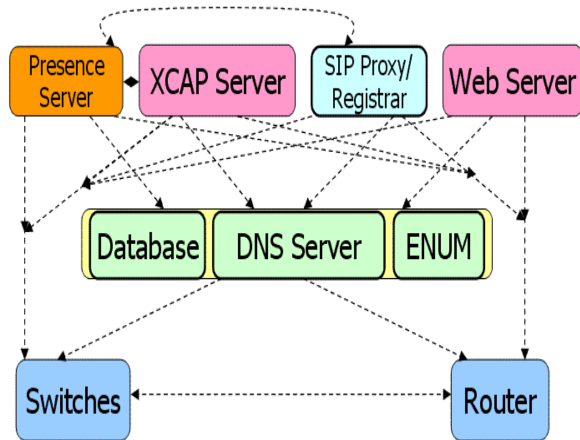

Figure 2 Different network components interacting

Consider a user alice@example.com 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 triggers the DYSWIS system to perform diagnosis which takes the following steps:

- The diagnosis node queries if any other node from caller's location has made a call to the user bob@destination.com. In this case, the location could mean from the same subnet, VLAN, access switch/access point or domain.
- The response can be that other nodes have recently made a call to same destination address or to the destination domain, not to same destination address or no node have made any call to destination address or destination domain. It should be noted that historical success or failure information is queried taking into account location of observed failure (hence using the topology) along with functional dependencies.
- Based on the response, the diagnosis node may request another node to send a SIP OPTION message to the destination address or it may request to make a call to the test node in the destination domain. This test is done both from location of original node which encountered a call failure as well as another location to see if this could be a problem specific to the caller's location. This will test the availability of remote SIP endpoint.
- Based on the response from the tests and historical information from other peer nodes, the location of failure is isolated. The next step is testing the proxy server in failure domain.
- The result of above rule based diagnosis is identification that the problem is present in local node or remote node or in local domain or remote domain and possibly in which component. This also depends on granularity at which rules (tests and queries) encode the dependencies.

Figure 3, 4 and 5; explain the above described approach of DYSWIS. The flow charts show the order of tests based on dependency graph encoded using the rules in the diagnosis nodes.

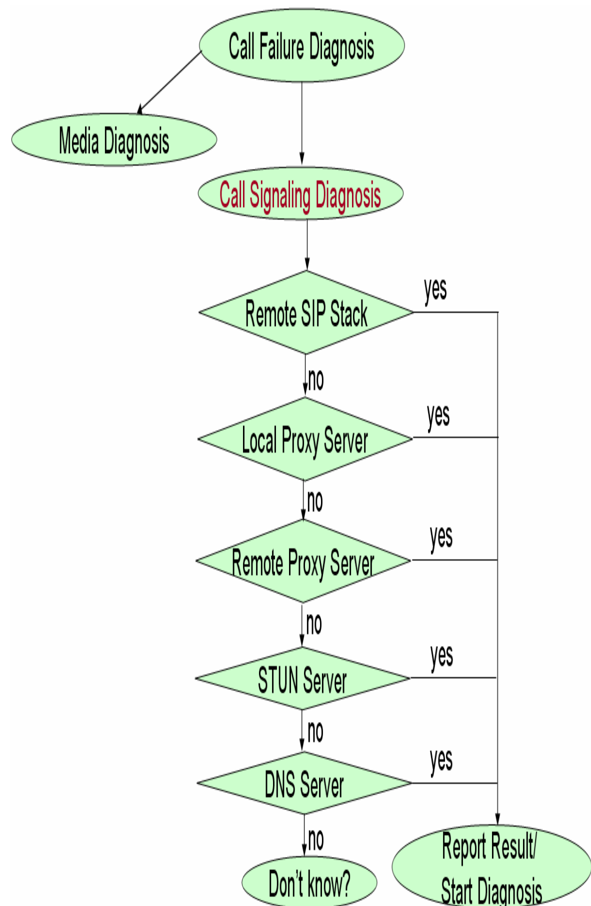

Figure 3 Call failure diagnosis

As we can see, Figure 3 shows the rules for 'call failure diagnosis' which in turn may trigger 'media failure diagnosis' (Figure 4) or 'proxy failure diagnosis' (Figure 5). The call failure diagnosis involves call signaling diagnosis which in turn involves testing of

remote SIP end point, local proxy server, STUN server and DNS server. These may further trigger diagnosis of network connectivity and availability of supporting protocols (services). The reports are stored and used to decide the order of queries for future failures. For example, if a call failure to the same destination is reported and diagnosis was already done for that failure, no more tests will be done.

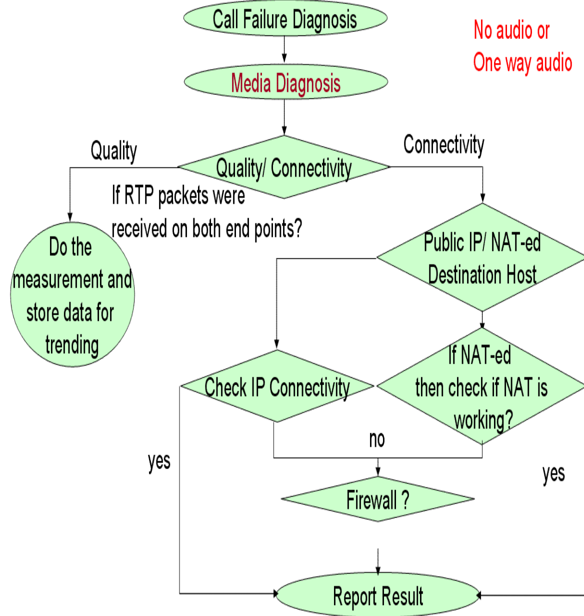

Figure 4 Media failure diagnosis

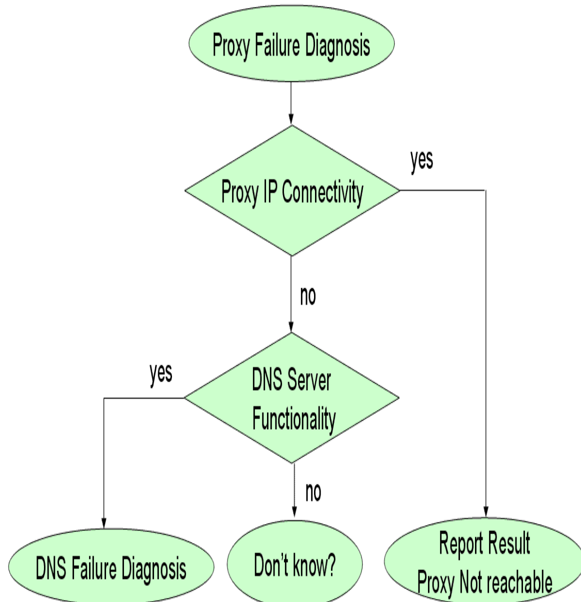

Figure 5 Proxy failure diagnosis

5. DYSWIS Implementation

In this section, we describe our implementation of DYSWIS diagnosis framework, overview of sensor nodes and diagnosis nodes, discovery of diagnosis nodes, request-response protocol used between DYSWIS nodes and finally, dependency representation. We also present an example diagnosis flow to illustrate the steps in diagnosis.

5.1 Diagnosis Framework

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

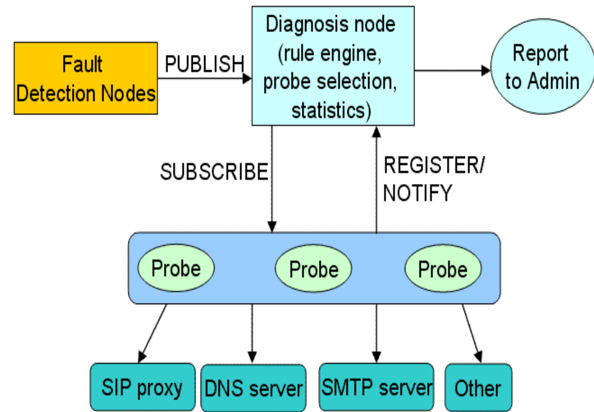

Figure 6 Diagnosis framework architecture

Figure 6 describes the work flow of DYSWIS framework. The lowest layer in Figure 6 shows network components, e.g., SIP Proxy, DNS server, SMTP server. There is an overlay of probe nodes which are capable of performing different types of tests. These nodes pre-register their location as well as testing capability to the diagnosis node. Diagnosis nodes are the intelligent nodes that have information about probe nodes, store historical failure information and store dependency rules encoded as tests and queries. When a DYSWIS sensor node detects a failure it updates the failure information by sending a SIP PUBLISH [14] request. This triggers a DYSWIS diagnosis node to start diagnosis which in turn may request the DYSWIS probe node using SIP SUBSCRIBE [14] to perform certain tests and submit the result back. The result is submitted back to the diagnosis node using SIP NOTIFY [14] request. The results of tests are used to determine further tests and also determine which nodes are suitable for performing which tests.

5.2 Type of DYSWIS Nodes

Now we describe about the various types of DYSWIS nodes and their interactions.

5.2.1 Sensor nodes

To detect failures in the network, the DYSWIS relies on sensor nodes which are active and passive nodes and are distributed in the network.

DYSWIS nodes passively sniff traffic and pro-actively report any failures. At the application layer, DYSWIS nodes can operate as "shadow" applications that access the same servers and services as the real application and thus help distinguish server and network faults from implementation failures and other conditions. For example, many embedded devices will silently ignore issues such as transient registration failures in SIP as they have no good way to report those to the user. A DYSWIS sensor node can monitor such failures by sniffing traffic (absence of 200 OK for SIP REGISTER) and alert the DYSWIS system before such failures result in loss of service, e.g., lost incoming calls because of registration failure.

DYSWIS sensor nodes also detect non-availability of service by conducting active tests. These nodes conduct periodic tests by emulating user action e.g., making a call. The nodes update the failure information once detected to a diagnosis node by sending a SIP PUBLISH message. These nodes are also called as probe nodes.

In our system which was implemented for VoIP network, we have implemented the following probes:

- SIP trace route using SIP OPTION message to determine application level path for SIP requests.
- SIP calls - INVITE tests – This can be used to test the proxy behavior as well as end point behavior.
- SIP OPTIONS
- NAT test – for one way audio
- Ping – IP layer reach
- Web server availability
- RTP test – jitter and loss for voice quality
- DNS tests – A, AAAA, SRV, MX record tests

5.2.2 Diagnosis Nodes

Diagnosis nodes are intelligent nodes in DYSWIS system and work based on a rule engine. We used the DROOL [11] 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. The SIP PUBLISH and NOTIFY requests received from sensor nodes update the historical failure database.

5.2.3 Request-Response Protocol

The DYSWIS nodes use a request response protocol to communicate among themselves, mainly:

- For notifying about detected failures to a diagnosis node.
- For requesting a probe to be run and results of the probe to be sent.

SIP PUBLISH and NOTIFY are used to report about failures and test results. SIP SUBSCRIBE is used to request active sensor nodes to perform tests. Other nodes can subscribe and get notified about diagnosis nodes available in the network at any point of time. We used NIST SIP [13] for implementing the request-response protocol for DYSWIS nodes.

5.2.4 Discovering the DYSWIS node

The probe and diagnosis nodes register themselves to with their capabilities to the DYSWIS system using SIP REGISTER request. This information (capability and location) is used to determine which node will do which tests. DYSWIS nodes can receive SIP NOTIFY and get updated about other nodes in the system. The end nodes can send SIP SUBSCRIBE to server and get information about diagnosis nodes using SIP NOTIFY.

5.3 Rule Engine – Dependency relationship

In this section, we explain different types of dependencies and we explain how we represented dependency in our implementation.

Dependency information can be represented at different levels. Functional dependency at service level e.g., SIP proxy depends on database (DB) service and DNS service. Instance level dependency are runtime bindings, e.g., the call which failed was using failover SIP server obtained using DNS SRV was running on a particular host. Dependencies can be represented at protocol level also, e.g., SIP depends on DNS. There can be vertical and lateral dependencies, e.g., applications can depend on other application layer services like SIP service depends on DB and DNS service. Applications also depend on transport layer services which in turn depend on network layer services. There can be topology based dependency:

e.g., calls from a particular domain depends on specific SIP server and specific switch and routers

5.3.1 Dependency representation

We represent dependencies using rules, using tests and queries using the DROOL rule engine framework. Queries are used to find information about past failures. Tests are probes and are used to conduct diagnostic tests.

```
rule "Call Failed"
   no-loop true
   when
              tm : FailureObject( failureType == TestObject.CALL )
   then
              CallFailureObject cfo = (CallFailureObject)tm;
              boolean result =cfo.doPreprocessing();
              if (result==true){
                        cfo.reportResult();
              }
              else {
                        cfo.setTestType(TestObject.SIPPROXY);
                        modify(tm);
              }
end

rule "Test SIP Proxy"
   no-loop true
   when
              tm : FailureObject( testType == TestObject.SIPPROXY )
   then
              CallFailureObject cfo = (CallFailureObject)tm;
              boolean result =cfo.doSIPProxyTest();
              if (result==false){
                        tm.setTestType(TestObject.DNS);
                        assert(tm);
              } else {
                        System.out.println
                        ("Starting SIP Proxy server diagnosis");
              }
end
```

Figure 7 Sample rules for dependency

Figure 7 above is a rule snippet of DROOL rule engine which we used. The rule says when an event 'Call Failed' occurs, test SIP proxy server, if SIP proxy is good, test the DNS server, otherwise, start diagnosis for DNS server. The "doPreprocessing" step queries for past failures, if the failure is already reported or diagnosed, it prevents the diagnosis to be started again.

6. Probe selection criteria

Some tests are computationally costlier and create more load on the network then others. There are also tests which are very specific to application types vs. a test which is more generic, e.g., SIP ping vs. IP ping. The order in which tests are done based on rules can have a significant impact on the speed of problem resolution. Most of the times the results from tests and sometimes statistical information help to determine which test must be done next. For example, an ICMP ping test and SIP ping test involve different costs on the host, the network components etc., and would result in vastly different results. An ICMP failure would remove the necessity to perform SIP level ping, whereas a failure at SIP ping would make the ICMP test as the next step. Additional factors that the probe selection algorithm considers are the location of the node which can perform the test with respect to the service which needs to be tested and with respect to service invocation which resulted in failure observed. For example, it may be useful to conduct a DNS test from two different subnets to ensure that problem of DNS failure is not local to the subnet and not local to the node itself.

7. Security Considerations

In our approach, nodes can ask other nodes to construct certain requests and send these requests to the target node. That way, new protocols can be supported without upgrading every DYSWIS node. However, such a system may lead to use of DYSWIS nodes as bot-nets that attack victims and become remote-controlled denial-of-service tools. Mechanisms must be built-into the architecture to prevent this. For example, rate limits and caching of results could ensure that each type of request, identified by its destination IP address and port, is only sent once within a time interval measured in minutes. The system also needs to avoid becoming a vector for mis-using address-based trust, allowing a third party to gain access to information that is restricted based on source IP addresses.

8. Future Work

In our implementation, we encoded dependency relationship as rules in the form of queries and probes; however, with networks growing in terms of components, services and protocols, there is a need to generate the dependency relation automatically using statistical mechanisms and using temporal correlation among failures detected. Secondly, we need instrumentation of applications to detect and report failures in order to trigger diagnosis. To detect failures without requiring software upgrade would require us to detect network failures using traffic analysis. This in turn would require specifying protocol details using a rule language to the traffic analyzer.

We identified requirement of a rule language for failure diagnosis. One of the tradeoffs in developing a rule language for diagnosis is simplicity vs. capability. An expert must be able to specify rules with ease without requiring much knowledge about a programming

language. However, this limits the functionality that can be expressed in a rule. The system needs to provide mapping between functionality of the system vs. the tools available to the expert. Providing a fixed mapping reduces the enhance-ability of the diagnosis system for new probes. A more scripting-based approach gives more flexibility but more complexity to the expert. A system which gives flexibility by taking external binaries/scripts and output of such binaries and scripts back to the rule language as well as provides a fairly high level way of representing knowledge may be good approach, a mix of XML and shell script style.

Finally, failure event correlation based on rules is another area of future work.

9. Conclusion

In this paper, we proposed DYSWIS system to automatically diagnose network failures and determine the root cause of failures and presented a reference implementation for a VoIP system. DYSWIS system can be implemented for any kind of network as long as probes can be defined, queries can be implemented and an expert can define the dependency rules based on existing probes and queries. We used the DROOL rule framework to represent the dependency information.

As a part of this work, we came up with requirement for a rule-based language which would meet the goals of a rule language for network diagnosis. 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.

10. 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", Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP.

[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. Proceedings. 17-18 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] Keller, A.; Blumenthal, U.; Kar, G., "Classification and computation of dependencies for distributed management", Fifth IEEE Symposium on Computers and Communications, 2000. Proceedings ISCC 2000. Volume, Issue, 2000, 78 – 83.

[7] Case, J., Fedor, M., Schoffstall, M., Davin, J., "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[8] Ping - http://ftp.arl.mil/~mike/ping.html

[9] Traceroute - http://tools.ietf.org/html/rfc1393

[10] Dig - http://linux.die.net/man/1/dig

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

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

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

[14] Jennings, C., Peterson, J., and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[15] 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.

[16] B. Gruschke.; "A New Approach for Event Correlation based on Dependency Graphs". In Proceedings of the 5th Workshop of the OpenView University Association: OVUA'98, Rennes, France, April 1998.

[17] Hasan, M., Sugla, B., Viswanathan, R.,; "A conceptual framework for network management event correlation and filtering systems"; Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management, 1999.

[18] 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.