VoIP Network Failure Detection and User Notification

Mark Karol
Avaya Labs
Room 1N-239
307 Middletown Lincroft Road
Lincroft, NJ 07738-1526
Tel: +1 732 852 2448
Email: mk@avaya.com

P. Krishnan
Avaya Labs
Room 2A-48
233 Mount Airy Road
Basking Ridge, NJ 07920-2311
Tel: +1 908 696 5208
Email: pk@avaya.com

J. Jenny Li
Avaya Labs
Room 2C07
233 Mount Airy Road
Basking Ridge, NJ 07920-2311
Tel: +1 908 696 5147
Email: jjli@avaya.com

***Abstract* - The impact of network failures can be minimized if users are promptly notified by appropriately designed applications. Specifically, for Voice-over-IP (VoIP) networks, an RTP/RTCP-based detection method can be used to rapidly distinguish between network congestion and network failures. Users and network managers can exploit this information in various ways, such as rapid network recovery or seeking application usage alternatives. In this paper, we present the main ideas behind these proposals, along with some analytical/simulation results, plus insights from a Linux-based implementation with its experimental results.**

Keywords – Voice over IP, Failure Detection

I. INTRODUCTION

Internet Protocol (IP) networks are characterized by packet delays, jitter, losses, and even occasional hardware/software failures (including those due to viruses and attacks). Yet, applications like Voice over IP (VoIP) are demanding underlying infrastructures to both be reliable and provide acceptable quality-of-service (QoS). Consequently, a lot of ongoing work is geared towards improving the reliability, performance, and QoS characteristics of IP networks. Much of the work involves monitoring network conditions and detecting network failures.

There are two broad categories of network monitors. The first type, distributed monitoring, typically assumes that the measurement instrumentation can be either distributed at different points in the network [1] [2] [3], or placed at the endpoints of the end-to-end path [4] [5] [6] [7]. The second way to monitor network conditions requires only one centralized observation point, which makes installation easier, but it takes longer to detect failures [8] [9].

With respect to failure detection, there are two common methods: (i) active injection of test probes into the network, and (ii) passive sniffing of existing network traffic. Many network measurement tools send periodic roundtrip echo probes to a distant host which then responds back to the sender. These probes are usually in the form of ICMP Echo packets; they are called NetDyn probes in [6] [10], Netnow probes in [11], and Fping [12] probes in Imeter [13].

To supplement these efforts to improve the reliability of IP networks, in this paper we examine the benefits associated with (i) *rapid failure detection* and (ii) *user notification*. These two items are particularly valuable for realtime applications such as VoIP.

In many cases, an application may be able to detect a network failure (or, in some cases, receive an explicit network notification regarding the failure) much faster than the time it takes to repair the failure (e.g., by restoration schemes). Current failure-detection methods typically notify only the network manager (for maintenance purposes); users do not receive alerts of network problems. We propose to inform users of network problems during VoIP sessions, which will allow users, for example, to decide whether to terminate their affected calls. The failure detection and notification may be specific to certain traffic sessions. Upon detection of a failure that the VoIP application knows will take some time to fix, it would be useful feedback to notify the end user about the failure. Such a feedback can take many forms: e.g., a voice or text message, or a light on the VoIP device that lights up saying, in effect, that a failure has occurred and "Network Restoration is in Progress." In other words, the network itself has put the user "On Hold." Various amounts of information could perhaps be sent to users, including, for instance, the anticipated length of time until restoration, if known (e.g., using a standard restoration procedure).

In this paper, in addition to proposing and discussing the general benefits associated with user notification, we also propose that a "keep-alive" signal be used for rapid failure detection. Rapid detection and notification of failures, long before the network is actually restored/repaired, can be exploited by users and network managers in various advantageous ways. For instance, users can decide whether to continue their calls ("on hold") or, instead, wait until the network is restored and make another call later. Also, network managers might take earlier recovery efforts, for example, to redirect existing and future calls and help prevent a bigger network outage. The key point is that once the user/manager is made aware of the network situation, they can decide what to do (or what not to do) long before the network recovers.

Without any support from network devices, a VoIP application might suspect a network problem exists if it does not receive packets on an RTP stream for a period of time (greater than the normal silence intervals). Note that not receiving any packets is different from receiving a few packets; the latter case may indicate bad voice quality that could be caused by congestion, lost packets, or other network problems. Network failures that require restoration schemes to kick in are different in that, after a short wait (which depends on the restoration scheme(s) used), one could expect to see voice quality that is comparable to before the failure. In this case, a user might be more inclined to wait until restoration happens and wait before continuing a dialog. Such network failures are often caused by network device or link failures.

Finally, we describe, for VoIP, a specific manner in which dynamic transmission of RTCP packets can serve the role of a keep-alive signal (for rapid failure detection). Normally, RTCP packets are primarily just used to provide feedback on the quality of the RTP data distribution. The key component of our method is a failure detection and notification box (DNB). More details of the box will be given in Section II. The box can be added to existing IP phones (endpoints in distributed monitoring), or placed in a network device to be shared by many endpoints like the centralized monitoring method. In effect, we use both sniffing and probing methods in the failure detection; new packets are injected and they are used in failure detection along with the existing traffic.

Network devices (e.g., edge routers and media gateways) can provide an added value in the proposed rapid failure detection. If these devices determine network problems, they can send notifications about it to end devices, which can then provide the user with feedback. The notification could be broadcast to all users, or, at the other extreme, sent to network devices that specifically maintain information about open sessions so that only those sessions are informed of the failure.

In Section II, we briefly describe the new techniques. In Section III, we discuss some analysis results and important tradeoffs. In Section IV, we present our experimental results, and our conclusions are in Section V.

II. VoIP FAILURE DETECTION AND NOTIFICATION METHOD

Currently, it is possible to provide voice quality feedback to a user participating in a VoIP session. Also, various metrics related to RTP (Real-Time Transport Protocol) traffic might be computed by an endpoint application and then presented to the user. However, current network devices do not automatically inform applications or users about network problems. These are mostly gleaned via concepts like timeouts. The emphasis in VoIP and other applications has been to deliberately hide the network and its problems from the user. In this section, we describe an alternative system design in which failures in a converged network are rapidly detected and then users and network managers are informed in a timely manner. We primarily describe the new system architecture in the context of VoIP user notification upon network failure. However, the general concept could also be used in other scenarios.

A. Description of the Technique

The new method, which can be either distributed or centralized, makes use of the existing VoIP infrastructure. The sender increases the number of RTCP packets when the number of RTP packet decreases (during silence periods), making sure that, for example, at least one RTP or RTCP packet is sent every T seconds. Specifically, additional (short) RTCP packets are injected at rate $R = 1/T$ during silent periods in the RTP stream. The receiver declares that a failure has occurred if neither RTP nor RTCP packets are received within a window of kT seconds ($k = 2, 3, \ldots$). The receiver then notifies the users who are involved in the sessions, plus, perhaps, users that might be affected if they were to attempt to set up a new call. Since the RTP/RTCP packets experience network jitter and loss, there is a chance that a failure will be wrongly declared (i.e., a “false alarm”). However, as soon as either an RTP or an RTCP packet is once again received, the receiver cancels its “failure announcement.” This could either correct a false alarm or announce restoration after an actual failure.

The failure detection and notification box (DNB) can be placed at any of four places: (i) inside the IP phone application, (ii) inside the RTP/RTCP stack of the IP phone, (iii) inside the edge device that connects the enterprise network to the internet, or (iv) as an independent box next to one or more IP phones or an edge device controlled by a call server. Missing RTP and RTCP packets can be monitored at the endpoint (i.e., distributed) or the call server (i.e., centralized). The centralized method uses the call server to detect existing calling sessions and their corresponding RTP/RTCP packets. It informs the end-user device of network failures after detecting missing RTP/RTCP packets. The distributed method uses an extension to the end-user IP phone to adjust the RTP/RTCP packet transmission rate and detect missing packets. The frequency of RTCP packets increases as the number of RTP packets decreases.

Figure 1 shows the setting of our method. It includes network components such as IP phones and edge routers. The DNB box can be implemented inside these devices or as an independent box.

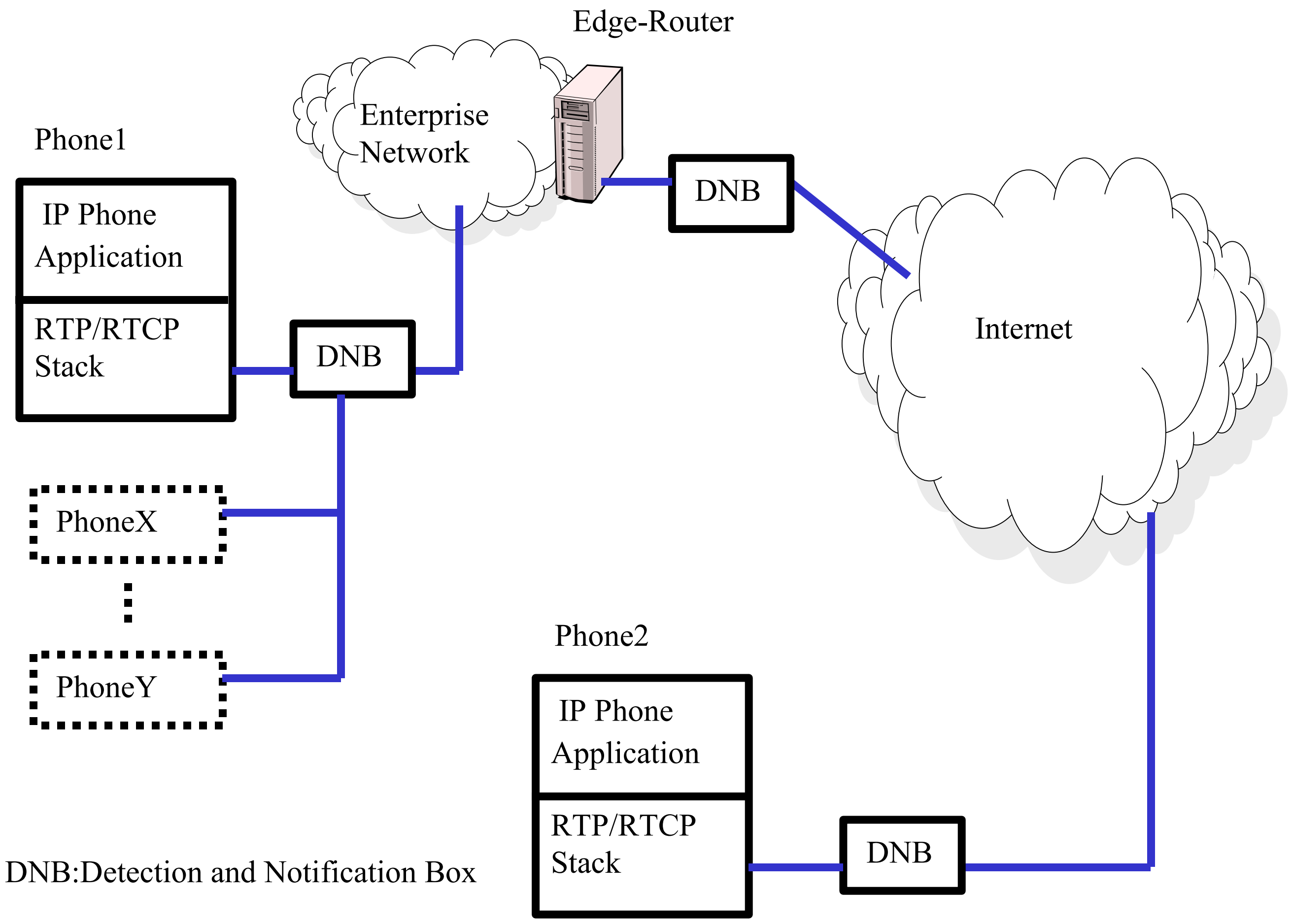

Fig. 1 A general architecture of failure detection and notification method

For a distributed implementation involving Phone1 and Phone2, we now briefly describe a typical failure detection and notification scenario. First, the DNB boxes of Phone1 and Phone2 inject some (short) RTCP packets into the session stream whenever no RTP packets are sent (i.e., during silence periods). Both DNB boxes monitor the incoming RTP and RTCP packets from the other party. If a DNB box does not receive either RTP or RTCP packets (for a time period *kT*), it declares that a network failure occurred and attempts to notify the other party. For example, if Phone1 detects a failure, it notifies Phone2 that a failure occurred on the path from Phone2 to Phone1. The user of Phone2 will be notified that no packets can be sent from Phone2 to Phone1, even if the path from

Phone1 to Phone2 is okay. If Phone2 also detects a failure, then a failure is detected that affects the traffic in both directions.

For a centralized implementation, a typical scenario would also make use of the edge device and its corresponding call server. Again, the DNB box can be implemented (i) inside the call server application, (ii) inside the call server RTP/RTCP stack, or (iii) as an independent box next to the edge device. When a call between Phone1 and Phone2 occurs, they both register their sessions with the call server. The server monitors physical, link, and IP layer connectivity, as well as RTP and RTCP traffic between Phone1 and Phone2. When it detects network failures, it notifies all affected sessions and endpoints. Even if the affected endpoints are not active, failure notification can prevent users from making unnecessary (failed) calls, which saves the users time and money.

B. Some Benefits and Applications of the Technique

Benefits of the new technique include reduced failure detection time and improved detection reliability. Its applications include early failure recovery by notifying network managers and giving users choice of actions.

Although network managers can improve reliability by adding alternate routing paths before outages occur, a more effective way to manage and ensure the converged network reliability is early detection and notification. The proposed technique enables round-the-clock failure notification. As the detection time increases, more users and connections are impacted and the cost of finding and repairing failed network system components rises (i.e., similar to the detrimental impact of a spreading virus). Early failure notification allows the network manager to fix the problem before it causes a bigger network breakdown.

Giving users a choice of actions is also important and valuable. After learning that the network connection itself is temporarily “on hold” (and will momentarily be restored), the user/application will make the better decision (e.g., lower delay) to just wait for the connection to be restored rather than close the connection and open a new one.

In addition, failure information can be used to inform a “smart router” of failures on a certain path so that a different path will be selected for future transmissions.

C. Some Implementation Issues

Figure 2 shows an example of the distributed method implemented at the end-user applications. A typical DNB box includes two major components: a RTP/RTCP sender for sending additional RTCP packets, and a failure detector for detecting and notifying failures to the other party. The RTP/RTCP sender is designed to increase the number of RTCP packets when the number of RTP packet decreases (during silence periods). If neither RTP nor RTCP packets are received during a time interval $kT = k/R$, the failure detector declares an incoming network failure. It sends a feedback to the RTP/RTCP sender to transmit an additional RTCP packet to the other party. Once the other party receives such a failure notification RTCP packet, it can notify its user that there is a failure in the sending of packets. If it does not receive the notification, a two-way network failure is detected and both users will be notified.

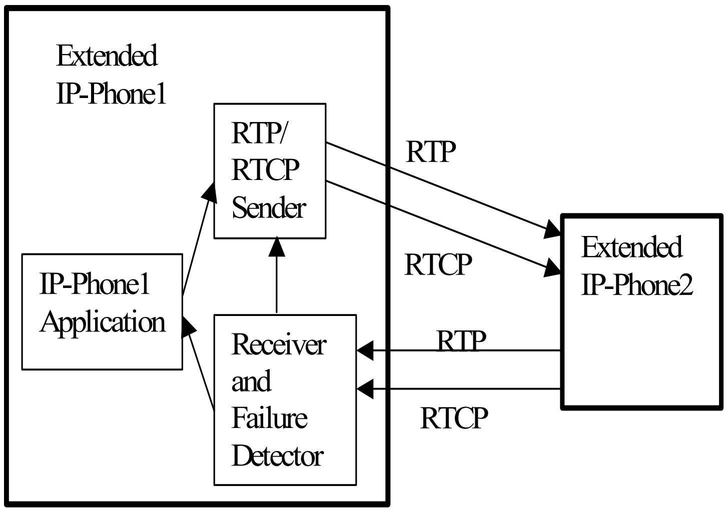

Fig. 2. Adjustable RTCP failure detection

Important implementation issues of the new technique include coordinating the number of RTP and RTCP packets sent by the RTP/RTCP sender component, and the failure detection criteria when none of the packets are received within a certain time interval. The key question is when and how to inject RTCP packets into existing RTP streams. Figure 3 shows the RTCP insertion mechanism.

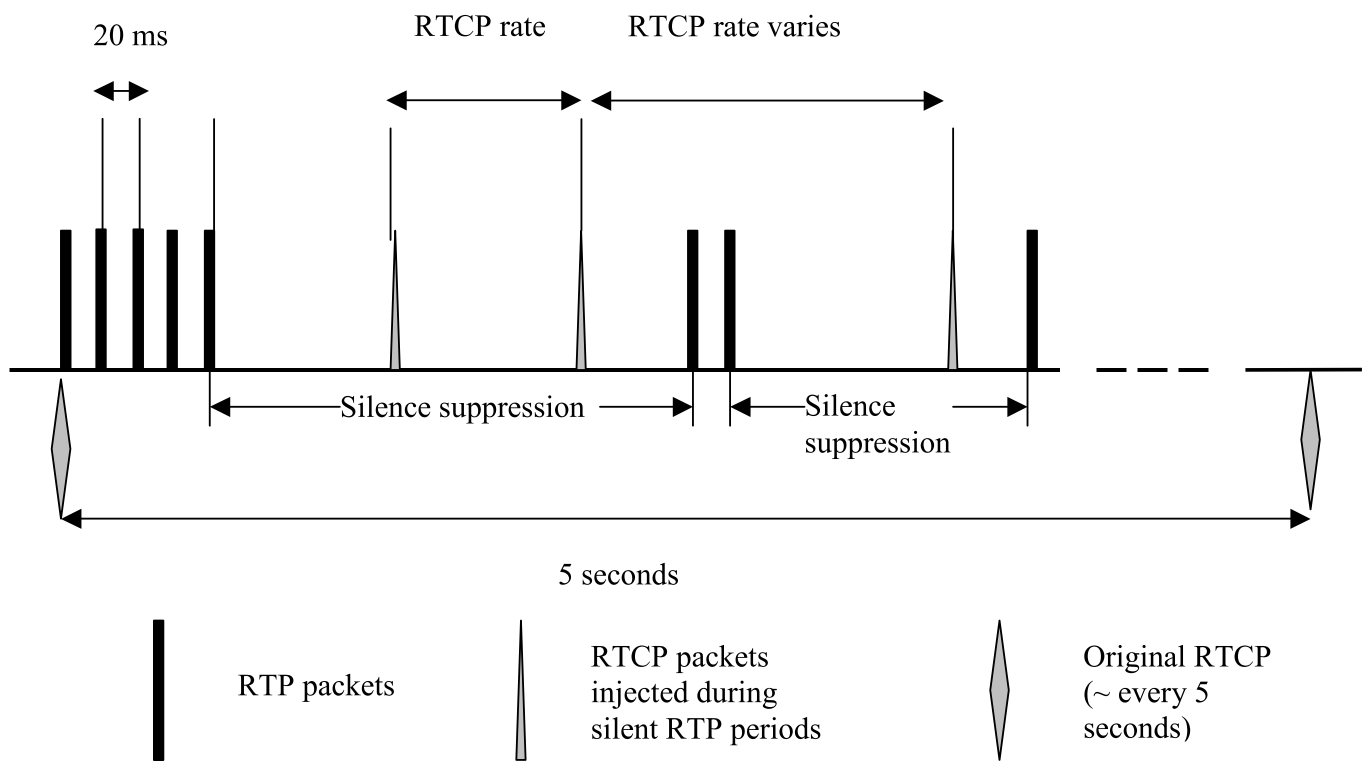

Fig. 3. RTCP packet injection mechanism

Another implementation option is to let the call session manager of a server keep track of the RTP and RTCP traffic on each session and thereby detect failures. An active end-user notification can be made from a network device (e.g., an edge router or media gateway) to an application that the user is interacting with, which then signals this notification (probably processed by rules) to the user. If these devices determine that a network problem has occurred (e.g., from network or reconfiguration messages that they receive), they can send notifications about it to end devices, which can then provide the users with feedback. The notifications could, for instance, be broadcast to all end devices. Alternatively, if network devices maintain information about open RTP sessions (which may require some capability like wire-speed string matching), then they can selectively notify only users in those sessions (perhaps by introducing packets into the RTCP sessions). In both cases, since a network problem (e.g., reconfiguration) may not necessarily affect a particular session, end applications must use such network notifications in conjunction with other factors (like not receiving packets for some time period) when deciding to notify the end user.

III. BANDWIDTH AND COST TRADEOFFS/ISSUES

In this section, we examine some preliminary tradeoffs between the rate of inserted RTCP traffic and the time it takes to detect failures. In general, the more RTCP packets are injected, the faster the failure detection will be (but with higher overhead).

The additional "keep-alive" RTCP packets can be sent as SDES packets [14, Section 6.4], with a source count (SC) of 0. The size of the data portion of such a packet is only four octets (excluding the 8-octet UDP and 20-octet IP headers). In comparison, the sender report RTCP packet is larger and requires statistics collection. Application-defined RTCP packets [14, Section 6.6] may also be considered for this "keep-alive" purpose. The RTP RFC [14, Section 6.2] also recommends against sending too many RTCP packets. The spirit in which this recommendation is made is adhered to in our proposal, since the additional injected RTCP packets are only sent when regular RTP packets are not (during silence periods, for example). Also, the overhead is minimal since we intentionally use very short RTCP packets for the keep-alive function. In particular, note that the SDES packet is smaller than an RTP packet, given the mandatory 12-octet RTP header. Furthermore, the SDES packets will be sent at a lower rate than the RTP packets. Hence, any pre-negotiated bandwidth limitations should not be exceeded.

For use in determining optimal RTCP rates in actual implementations, we now present a formula that relates the RTCP rate to RTP rate, failure detection time, and overhead (including UDP and IP headers). We use the following notations:

i. Injected RTCP Packets: Rate = R packets/sec; Length = 32 (i.e., 4+8+20) octets

ii. RTP Packets: Rate = r packets/sec; Length = header_length (RTP+UDP+IP) + c = 40 + c (c denotes the payload length, which is a function of the codec)

iii. Regular RTCP Packets: Rate = 1/5 packets/sec (which is the maximum recommended rate in [14]); Length = header_length + w = 28 + 52 = 80 (w denotes the length of the sender statisitics report)

iv. Percentage of silence during a call: D

With this notation, we can represent the total additional injected RTCP overhead as: $32RD/[r(1 - D)(40 + c) + 32RD + (28 + w)/5]$. Also, the failure detection time is k/R, where k is the number of missing RTCP intervals before a failure is declared. Suppose, for instance, (i) RTCP packets are injected every 500 ms during silence periods (R = 0.5), (ii) the codec generates RTP packets every 20 ms (r = 50) with 160 octets of payload, and (iii) the call is silent 50% of the time (D = 0.5) – i.e., while the other participant is talking. Then the injected RTCP overhead is only

0.16 percent and the failure detection time is 1.5 seconds if we select k =3 as the threshold for declaring a failure. The overhead is so small that it is almost negligible to overall network traffic.

IV. EXPERIMENTAL RESULTS

We implemented a version of our new failure detection and notification method on an IP phone application. Figure 4 shows the setting and the components of the implementation. It includes two IP soft phones and one router, all of which ran on Linux machines. We installed NISTnet[15] on the router to emulate network failure and QoS (delay, loss, jitter) impairments. We implemented the failure detection and user notification components on the phone application. The sender of the application was modified to insert RTCP packets during silence periods, and the receiver was modified to wait for additional RTCP packets during silence periods. A new component, detector and notifier, was added to the application to decide when failure occurs and to signal the failure to the user.

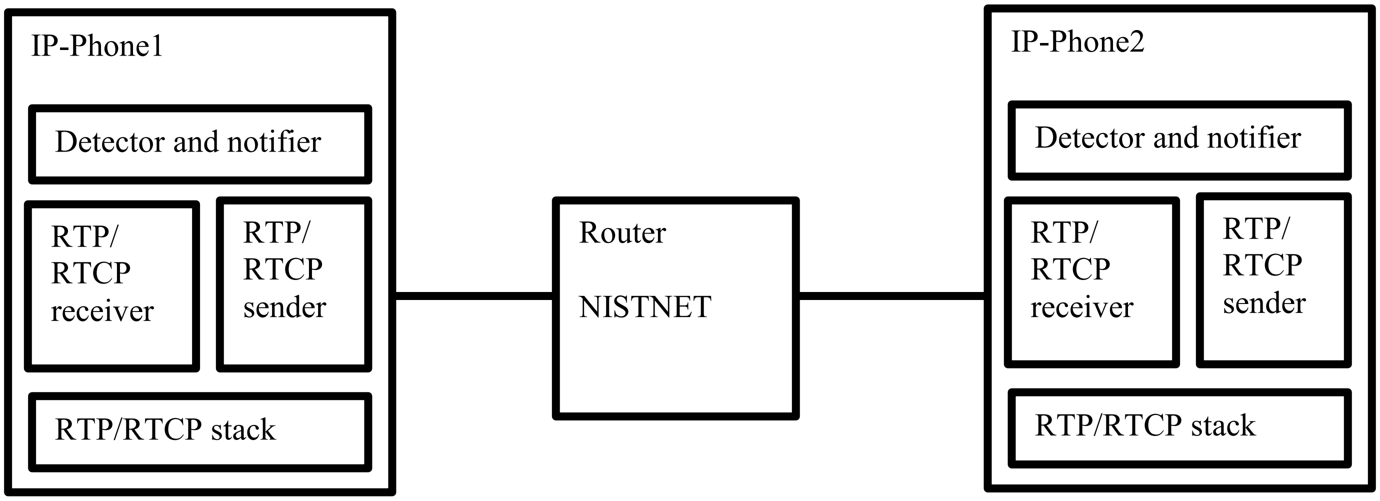

Fig. 4 Experimental setting of failure detection and notification method

On this implementation platform, we carried out a set of failure detection and notification experiments. We designed two kinds of experiments to study the failure detection times and the false-alarm probabilities:

A. In this set of experiments, we emulated network failures and measured the failure detection times. We used k = 4; i.e., a detection time equal to four times the injected RTCP intervals ($1/R$). For each RTCP rate, we repeated the detection experiment 10 times to obtain an average detection time and standard deviation. Table 1 shows the results. In each experiment, the detection time is dependent on whether the failure occurs during a talkspurt (when the RTP packets were spaced 20 msec apart) or during a silence period (when the RTCP packets were spaced further apart). Thus, as expected, the average detection time was close to (and less than) *k/R*.

RTCP rate (ms) (*1/R*)	Average Detection Time (ms)	Deviation (ms)
100	368	47.1
200	719	82.8
500	1735	191
1000	3222	510
2000	7050	736

Table 1: Experiments A

B. In this set of experiments, we examined the effects of packet loss on the probability that a false-alarm failure is declared. The experiment was repeated 10 times for each value of *k*. Table 2 lists the number of times that a false alarm occurred during a 10-minute call. During silence periods, RTCP packets were injected every 200 ms. A (false-alarm) failure was declared if *k* successive RTCP packets were dropped by the network and if the corresponding silence period (i.e., no transmission of RTP packets) lasted more than *k* times 200 ms.

Packet Loss (%)	*k*	Number of False Alarms
5	2	10
5	3	0
5	4	0
10	2	10
10	3	0
10	4	0

Table 2: Experiments B

These and our other experiments have shown the effectiveness of the proposed failure detection and notification method. In future work, we will include the effects of packet delay jitter in the experiments. However, we believe that packet loss is the major contributor to false alarms; we suspect the impact of jitter will be negligible. Typically, the failure detection intervals (*k/R*) will be much greater than the delay jitter experienced by RTP and RTCP packets. It is unlikely that jitter alone will cause the detection interval to pass without reception of any RTP or RTCP packets.

V. CONCLUSIONS

In this paper, we proposed a method for rapid network failure detection for VoIP. It reduces failure detection time by coordinating the sending frequency of RTP and RTCP packets. There are three key aspects to this new design. First, it exploits the fact that detection times today are often faster than network restoration times. Second, it recognizes that there typically is a significant difference between the information available at network devices and the information that is actually conveyed to users/applications. Although this difference is often for good reason (e.g., not overburdening users/applications with unimportant information), there sometimes are advantageous ways for users/applications to exploit information about the network conditions (which is the third key aspect of our proposal). Third, it exploits the notion that the applications that are using the network currently are most interested in knowing about its state, and so the application can deduce this information from packets flowing in the session and report it to the user.

In the future, we plan to experimentally determine various failure detection speeds and capacities. Also, it would be interesting to measure the restoration times in real networks and compare with the detection and reaction times attainable with our design.

REFERENCES

[1] P. Francis, S. Jamin, V. Paxson, L. Zhang, D.F. Gryniewicz, and Y. Jin, "An Architecture for a Global Internet Host Distance Estimation Service," *Proc. IEEE INFOCOM'99*, March 1999.
[2] S. Jamin, C. Jin, Y. Jin, Y. Raz, Y. Shavitt, and L. Zhang, "On the Placement of Internet Instrumentation," *Proc. IEEE INFOCOM'2000*, March 2000.
[3] W. Theilmann and K. Rothermel, "Dynamic Distance Maps of the Internet," *Proc. IEEE INFOCOM'2000*, March 2000.
[4] V. Jacobsen, "pathchar – A Tool to Infer Characteristics of Internet Paths," April 1997, ftp://ftp.ee.lbl.gov/pathchar.
[5] A. B. Downey, "Using pathchar to Estimate Internet Link Characteristics," *Proc. ACM SIGCOMM'99*, Aug. 1999.
[6] J.-C. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *Proc. ACM SIGCOMM'93*, Sept. 1993.
[7] K. Lai and M. Baker, "Measuring Bandwidth," *Proc. IEEE INFOCOM'99*, March 1999.
[8] http://www.protect-me.com/anm/reviewer_guide.pdf.

[9] D. Andresen, S. Kota, M. Tera, and T. Bower, “An IP-Level Network Monitor and Scheduling System for Clusters,” *Proc. 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02),* Las Vegas, June, 2002.
[10] J.C Bolot, “Characterizing End-to-End Packet Delay and Loss in the Internet,” *Jour. High-Speed Networks*, 2(3):305-323, 1993.
[11] NetNow Probing Program, http://www.merit.edu/ipma/netnow/daemon/, 1999.
[12] Fping Program, ftp://networking.stanford.edu/pub/fping/, 1997.
[13] J. Sedayao and K. Akita, “LACHESIS: A Tool for Benchmarking Internet Service Providers,” *Proc. 9th System Administration Conference(LISA95)*, Sept. 1995.
[14] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Application for Real-Time Applications,” IETF RFC 1889, January 1996.
[15] NIST Net, http://snad.ncsl.nist.gov/itg/nistnet/.