Characterizing and detecting relayed traffic: A case study using Skype

Kyoungwon Suh, Daniel R. Figueiredo, Jim Kurose, Don Towsley
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
Email: {kwsuh,ratton,kurose,towsley}@cs.umass.edu

UMass Computer Science Techincal Report 2005-50

July-11-2005

(Updated on July-13-2004)

Abstract

Networked application developers have recently started to make use of relay nodes – application instances that also act as bridges between pairs of hosts running the same application. Relay nodes can bring costs to both users and network operators, at least in terms of increased bandwidth consumption. An interesting problem is to characterize the nature of relayed traffic and to detect its presence in the network. We focus on characterizing and detecting relayed traffic generated by multimedia applications. As a case study we use Skype, a popular voice over IP application that uses relays. Our technique relies solely on flow-level characteristics rather than on application- or protocol-specific information. Using two different controlled experimental environments we generate and collect a large amount of Skype traffic. In order to characterize the nature of relayed traffic, we introduce a few metrics. These metrics together with the results obtained from the characterization, are then used to detect Skype-relayed traffic from a packet trace collected at the access point of a large network. We show that the metrics proposed can be used to reliably detect Skype-relayed traffic.

I. Introduction

Networked applications are using network resources in increasingly ingenious ways in order to achieve high scalability and circumvent security limitations. One such example is the emergence of relay nodes – application instances that not only provide application services to local users, but also act as bridges between remote nodes running the same application. The use of a relay allows two nodes that could not otherwise communicate (e.g., due to firewall or NAT [1] restrictions) to do so, and can improve the quality of the communication between two nodes by avoiding congested or faulty paths [2].

Although relay nodes are useful to application designers, they have disadvantages from the perspective of users and network operators. A node chosen to act as a relay must bear the cost of relaying traffic. This cost is evident to users in the form of slower communication (due to bandwidth use by the relayed traffic) or financial costs, if payment for network access is a function of bandwidth usage. Network operators and small ISPs may also see drawbacks, as relayed traffic can increase the amount of traffic entering and leaving their network. For example, a machine running Skype [3] in our campus relayed more than a Gigabyte of voice traffic (in total) over 20 days.

Given the increasing use of application-level relay nodes and their potential costs to users and network operators, it is of interest to characterize the nature of relayed traffic and to detect its presence in the network. Network operators would probably like to know if traffic is being relayed through hosts that belong to their network. While there is an increasing number of applications that make use of relay nodes, we will find it useful here to distinguish between multimedia applications (where data flow continuously and have a moderate bit rate, e.g., greater than 15 kbps), and interactive applications (where data flow sporadically in short bursts and generally have a lower average bit rate, e.g., less than 5 kbps). Examples of multimedia applications that use relays are Skype and End System Multicast (ESM) [4], while examples of interactive applications are Internet Relay Chat (IRC) [5], MSN Messenger, and SSH. We will focus on characterizing and detecting relayed traffic generated by multimedia applications, as these applications are more bandwidth demanding and are likely to be more costly to users and network operators.

In this paper, we characterize relayed traffic generated by a popular voice over IP application, namely Skype [3], and propose a methodology for detecting Skype-relayed traffic. We propose the use of several metrics to characterize the nature of multimedia relayed traffic. In order to obtain relayed traffic, we create two different experimental environments that are used to generate and collect a large amount of Skype-relayed traffic. This data is characterized using the different metrics proposed. We use the results of the characterization to propose a methodology for detection of Skype-relayed traffic. Detection is performed by setting thresholds for the metrics. By tuning the thresholds, the desired balance between true positives and true negatives can be obtained. Finally, we apply the detection methodology to a large aggregate traffic trace collected at the access point of our university. Our results show that the metrics considered for characterizing relayed traffic can reliably detect Skype-relayed traffic.

An important consideration when designing a traffic classification method is the use of application- or protocol-specific information, such as well-known port numbers. Since applications are increasingly becoming more flexible and diverse, new methods for traffic classification no longer rely on this information [6]. As in such approaches, our methodology for characterizing and detecting relayed traffic also does not rely on application- or protocol-specific information. However, our methodology does take into consideration the fact that applications are transmitting multimedia traffic.

The evaluation of our detection methodology is also a challenge, since it is not trivial to know when traffic is indeed being relayed. In order to obtain a benchmark for Skype-relayed traffic, we propose a heuristic for identification of Skype traffic that uses Skype-specific information. Using this information, we obtain the true set of Skype-relayed traffic. Although of independent interest, the heuristic is used only for evaluation of the detection methodology.

Some recent efforts have addressed different aspects of the relay detection problem. For example, the use of the ON-OFF characteristics of interactive applications, such as SSH, has been proposed to detect relayed traffic [7, 8]. The temporal order of flow start times has also been suggested as a mechanism to identify traffic traversing a network node [9]. However, none of these efforts have investigated multimedia traffic relays. To the best of our knowledge, this is the first work to characterize multimedia relayed traffic and to propose a methodology for its detection.

The remainder of this paper is structured as follows. In the next section we define relayed traffic and present the metrics used for its characterization. In Section III, we discuss our controlled experimental environment and characterize Skype traffic using the proposed metrics. Section IV presents the payload-based Skype traffic identification heuristic. In Section V, we evaluate the performance of detecting Skype-relayed traffic by analyzing an aggregate traffic trace collected at our university's gateway. Section VI presents a discussion of the related work. Finally, Section VII concludes the paper with a summary of our contributions and discussion of future work.

II. Problem Definition

Relayed traffic is traffic that is generated by a given end-host, and then passed through one or more end-hosts ("relays") before reaching its final destination. Figure 1 shows an example of relayed traffic. End-host n_1 generates traffic that first passes through end-host n_r before reaching its final destination, end-host n_2. From the perspective of the transport layer, there are two bidirectional connections in this example: one between n_1 and n_r and another between n_r and n_2. As we will see shortly, these connections may differ in type (e.g., one connection may be TCP and the other UDP), duration, and throughput.

A relay node may or may not be interested in the data traffic being relayed, and may or may not perform transformations on the relayed data. For example, node n_r may be watching the streamed video generated by node n_1 that is then forwarded to node n_2. On the other hand, node n_r might simply act as a bridge between node n_1 and n_2, receiving and sending packets. Many different types of transformations can be made, ranging from simple transformations such as transport protocol change (e.g., UDP to TCP), application-level header modifications, and fragmentation/composition of data packets, to more complex operations ,such as data compression/decompression, and CODEC changes. In principle, a relay node can perform any kind of transformation on the relayed data. In practice, however, current relay nodes perform only simple transformations, most likely in order to minimize the computational and communication load placed on relay nodes.

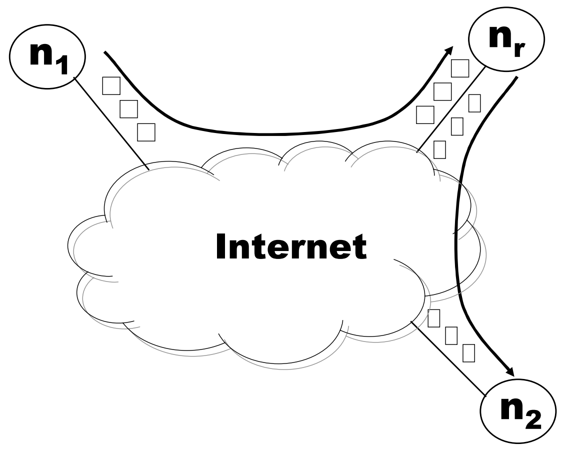

Fig. 1. Example of relayed traffic between end-hosts n_1 and n_2

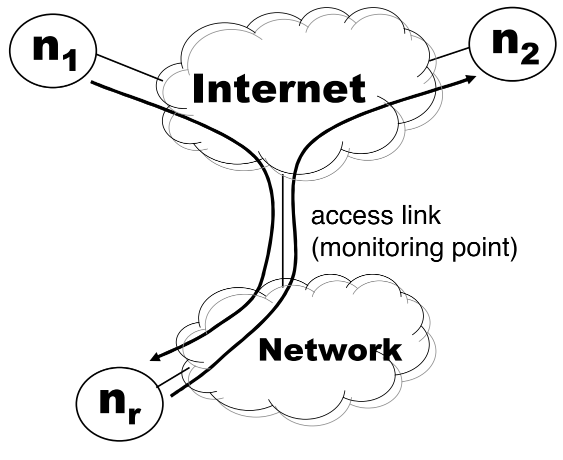

Fig. 2. Network scenario under consideration. End-host n_r is used as a relay by two end-hosts outside the network

In this paper, we take the perspective of the operator of a large network, who is monitoring incoming traffic at an access link, as illustrated in Figure 2. Our goal is to determine whether traffic is being relayed through some end-host belonging to the network, using only IP and transport header information collected by the monitor to make this decision. In general, this problem is hard and we will focus on relayed traffic generated by Skype. We will use the fact that Skype-relayed traffic is voice traffic (which poses constraints on maximum delays and minimum bit rates) but will not use application specific information such as well-known port numbers or distinct packet sizes, nor will we examine packet payloads.

Thus far, we have only informally defined a relay and the traffic being relayed; let us now make this definition more precise. Intuitively, we know that a relay takes an input “flow” of data and forwards the data to its final destination. But what is meant by a “flow”? In many measurement studies, a “flow” is defined as a sequence of packets with the same 5-tuple (source and destination IP addresses, source and destination port numbers, protocol number) [10, 11]. But this definition alone will not suffice for our purposes for several reasons. First, we will need to add the notion of a direction to a “flow”(e.g., source-to-relay, or relay-to-final-destination). Second, we will need to add a requirement that not “too much” time elapse between adjacent packets in a “flow” - a requirement often made in measurement studies (e.g., a 60 second maximum inter-arrival time between adjacent packets in a flow [10, 11]). This requirement will help us distinguish among distinct Skype calls that are transferred between a source and a relay node within the same 5-tuple-connection. For example, Figure 3 shows two voice calls (the first call starting at approximately 200 seconds and ending at approximately 400 seconds, and the second call starting at approximately 800 seconds) that are relayed in the same 5-tuple flow between a given source and a given relay. Finally, since it is voice calls that are being relayed, we want to distinguish between signaling traffic between a source and a relay (or a relay and a destination), and the actual voice traffic being relayed. Thus, we will want to associate both a minimum and a maximum bitrate to Skype-related traffic.

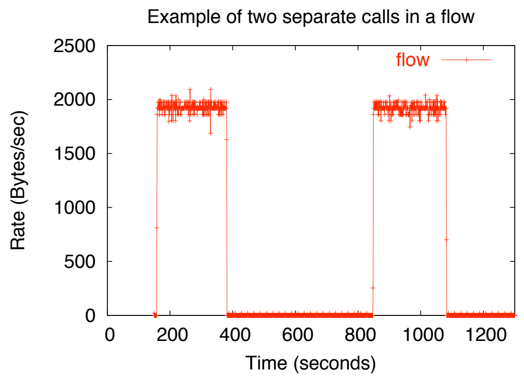

Fig. 3. Example of two bursts corresponding to two different voice calls within the same flow

Given these consideration, we will use the notion of a *burst of packets* (to avoid confusion with the colloquial use of the word "flow") in order to characterize relayed traffic. A burst of packets is a contiguous piece of a flow and is defined as a sequence of packets that has a minimum average data rate (e.g., at least 10 kbps) and a minimum duration (e.g., at least 30 seconds). In order to detect the start time and end time of a burst of packets we use a simple change detection algorithm. In particular, we adopt an EWMA (Exponential Weighted Moving Average) to keep track of the data rate associated with a flow. The EWMA of a given flow is defined as follows:

$$A_i = (1-\alpha)A_{i-1} + \alpha I_i \tag{1}$$

where I_i represents the average data rate of the i-th second of the flow and A_0 and I_0 are set to zero. Note that the EWMA is updated every second since the start of a flow. A burst starts if the EWMA goes above a certain threshold ($A_i > R_1$) and ends when the EWMA goes below another threshold ($A_i < R_2$). In order to detect the start of a burst we use a more aggressive parameter for the EWMA (large α). However, we use a more conservative parameter to detect the end of a burst (smaller α). In particular, we use $\alpha = 0.75$ and $\alpha = 0.15$ for the two cases, respectively. Each flow has a single EWMA value and its parameter, α, is changed as soon as the start/end of a burst is detected.

Each burst i contains a sequence of packets and has several quantities of interest:

- N_i: number of packets in burst i
- $T_i^j, j = 1, \ldots, N_i$: timestamp of the j-th packet of burst i
- $Z_i^j, j = 1, \ldots, N_i$: size in bytes of the j-th packet of burst i
- $S_i = T_i^1$: start time of burst i in seconds
- $E_i \geq T_i^{N_i}$: end time of burst i in seconds
- $Y_i^k, k = 1, \ldots, \lceil E_i - S_i \rceil$: total number of bytes sent during the k-th second of burst i (mathematically, $Y_i^k = \sum_{l=1}^{N_i} I(T_i^l - S_i \in [k-1, k))Z_i^l$, where $I(\cdot)$ is the indicator function, returning 1 when its argument is true and 0 otherwise)
- $B_i = \sum_{j=1}^{N_i} T_i^j$: total number of bytes carried by burst i
- $R_i = B_i/(E_i - S_i)$: average data rate of burst i

Note that it suffices to monitor TCP/UDP packet headers to construct flow and burst structures.

We will use the notion of a burst to characterize and detect relayed traffic. We will assume that in order for two bursts to be carrying relayed traffic they must have opposite directions (one entering, the other leaving the network), have the same end-host (IP address) within the network being monitored and have different end-hosts (IP address) outside the monitored network. Although the second assumption is not necessary, as traffic can be relayed through more than one end-host within the monitored network, we consider it for simplicity and computational feasibility. Also, note that Skype uses at most one relay node. The problem of relay detection is to determine if two bursts that could be carrying relayed traffic (e.g., candidate bursts) are in fact carrying relayed traffic.

In order to detect relayed traffic we will focus on a few statistical metrics involving a pair of bursts. Let i and j denote two bursts. We will consider the following metrics:

- $S_{i,j} = |S_i - S_j|$: the difference between the start times of bursts i and j
- $E_{i,j} = |E_i - E_j|$: the difference between the end times of bursts i and j
- $B_{i,j} = B_i/B_j$: the ratio between the number of bytes carried by bursts i and j
- $X_{i,j}$: the maximum cross correlation between time series Y_i and Y_j

Here $X_{i,j}$ is defined as $\max_{d=-N_i,\ldots,0,\ldots,N_i} x_{i,j}(d)$, where $x_{i,j}(d)$ is the cross correlation between time series Y_i and Y_j at lag d [12]. For completeness, this is defined as:

$$x_{i,j}(d) = \frac{\sum_k (Y_i^k - R_i)(Y_j^{k-d} - R_j)}{\sqrt{\sum_k (Y_i^k - R_i)^2}\sqrt{\sum_k (Y_j^{k-d} - R_j)^2}} \tag{2}$$

The metrics defined above will help us assess if two candidate bursts indeed carry relayed traffic. In particular, consider two bursts corresponding to Skype relayed traffic (e.g., voice over IP). In this case, the start time difference and end time difference of the bursts should be very small, as packets cannot be stored at the relay node for long periods of time. Moreover, since Skype does not perform complex transformations on the relayed traffic, the burst size ratio of the bursts should be close to one and the time series of the two bursts should have a very high degree of correlation. These observations will allow us to detect Skype relayed traffic.

III. Characterization of Skype-relayed Traffic

In this section we characterize Skype-relayed traffic using data collected from two different controlled experiments. In the first experiment, we control the two nodes that are used to make a relayed Skype call. This allows us to generate relayed traffic with different characteristics, such as different transport protocols and different packet loss rates. In the second experiment, we control the relay node used to relay traffic between two Skype users. This provides us with real relayed traffic, generated by real Skype users. The goal of both experiments is to collect a large amount of Skype-relayed traffic and then characterize this data using the metrics presented in the previous section.

A. Experiment I: Relayed traffic generated between two controlled Skype nodes

In this first experiment we control the two hosts running Skype that are used to make a relayed Skype voice call. By using a firewall to block packets between the two hosts we force Skype to use a relay node. This relay node is not under our control and is chosen by Skype. Moreover, by adequately configuring the firewall, we can control the transport protocol used by Skype both to and from the relay node. Since Skype can use both TCP and UDP to deliver voice packets [13], we have four possible pairs of transport protocol combinations: UDP_in-UDP_out, TCP_in-TCP_out, UDP_in-TCP_out, and TCP_in-UDP_out, where "in" and "out" represent traffic flowing to and from the relay node, respectively. Finally, we also introduce artificial packet loss on the path between one of the end-hosts and the relay node. By controlling the packet loss rate on one of the paths, we obtain Skype-relayed traffic under different loss regimes. We used dummynet [14] to generate Bernoulli losses. The experimental setting discussed here is illustrated in Figure 4. Note that data is collected by a host connected to the same hub where the two controlled nodes are connected. Using tcpdump [15] we capture all packets sent from SC_1 to SR and all packets sent from SR to SC_2. Note that we capture packets sent from SC_1 *before* losses are artificially introduced.

The above experimental setting was used to make hundreds of 3 minute Skype-relayed calls (we automated the call setup/tear down procedure). Throughout the experiment, several different relay nodes were used and although most of them were located in the United States, some relay nodes in Europe and Asia were also used. Table I summarizes the parameters and results from the experiment.

Using the data collected we characterize Skype-relayed traffic using the metrics described in the previous section. This characterization provides a benchmark for Skype-relayed traffic that can then be used as guidelines for detection of Skype-relayed traffic. As we next discuss, Skype-relayed traffic has particularities that can be used to identify it without resorting to any application-specific information.

The results are presented for each pair of transport protocol and for different packet loss rates. We start investigating the start time difference between two bursts of Skype-relayed traffic (recall $S_{i,j}$, where in this case i and j are Skype-relayed bursts). Figures 5 and 6 show the cumulative distribution of the start time difference. Note that all relayed bursts have a start time difference of less than 5 seconds. In fact, the vast majority of the relayed

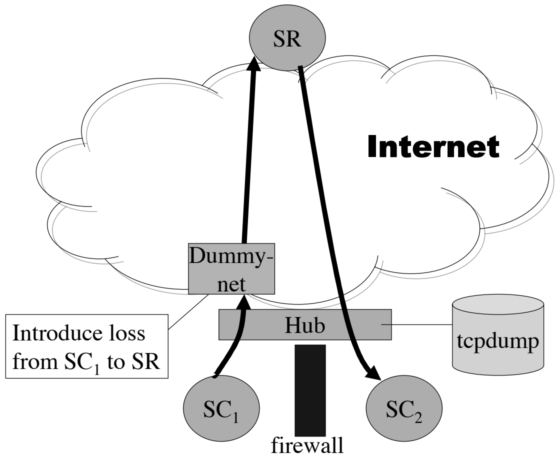

Fig. 4. Configuration of controlled experiment I

TABLE I
PARAMETERS AND RESULTS FROM CONTROLLED EXPERIMENT I

Parameters	**Value**
Transport protocol type	UDP_in_UDP_out, UDP_in_TCP_out, and TCP_in_TCP_out
Packet loss probability	0.00 and 0.05
Duration of each call	approximately 3 minutes
Number of UDP_in_UDP_out calls	505
Number of UDP_in_TCP_out calls	269
Number of TCP_in_TCP_out calls	307
Total number of relays used	1081

bursts (i.e., 99% of them) has a start time difference of less than 3 seconds. We also observe that introducing a packet loss rate of 5% has little effect on the start time difference.

Figures 7 and 8 show the cumulative distribution for the end time difference between two bursts of Skype-relayed traffic (recall $E_{i,j}$, where in this case i and j are Skype-relayed bursts). The characterization of the end time difference is similar to the start time difference when no artificial packet loss is introduced. Note that all end time differences are less than 5 seconds. However, for the case with 5% packet loss end time differences are longer, especially for the TCP_in_TCP_out case. This increase is due mainly to packet retransmissions at the end of the voice call and the loss of TCP FIN packets, which are responsible for gracefully terminating the TCP connection. In any case, all end time differences between two bursts of Skype-relayed traffic are less than 16 seconds.

Figures 9 and 10 show the complementary cumulative distribution of the maximum cross correlation between two bursts of Skype-relayed traffic (recall $X_{i,j}$, where in this case i and j are Skype-relayed bursts). We observe that the transport protocol pair significantly impacts the maximum cross correlation. As one might expect, the UDP_in_UDP_out combination yields the best maximum cross correlation, as the traffic in both connections is not shaped by any congestion control or flow control mechanism. In particular, note that 85% of the relayed traffic has a maximum cross correlation greater than 0.9 in this case. A similar result is obtained for the UDP_in_TCP_out combination, although with a slightly lower maximum cross correlation. Finally, the TCP_in_TCP_out yields an even lower maximum cross correlation, especially under the 5% packet loss rate. This is expected, as the first TCP burst will contain many packet retransmissions while the second burst will not. In any case, the maximum cross correlation is still surprisingly high, even in the TCP_in_TCP_out. In fact, 95% of all Skype-relayed traffic has a maximum cross correlation of at least 0.37, regardless of transport protocol combination and packet loss rate (not shown in graphs).

Finally, we investigate the burst size ratio between two bursts of Skype-relayed traffic. Figures 11 and 12 show the cumulative distribution of the burst size ratio (recall $B_{i,j}$, where in this case i and j are Skype-relayed bursts). Note that for the case where no artificial packet loss is introduced, the burst size ratio of all Skype-relayed traffic

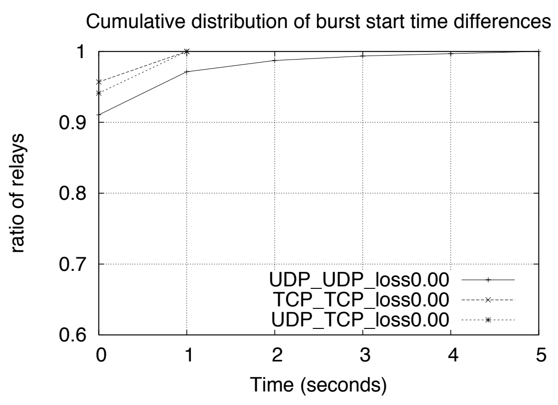

Fig. 5. CDF of burst start time difference

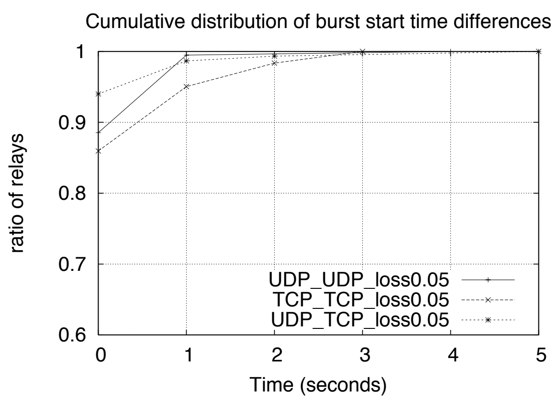

Fig. 6. CDF of burst start time difference

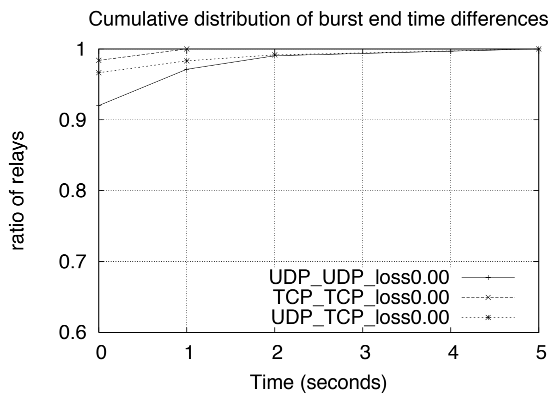

Fig. 7. CDF of burst end time difference

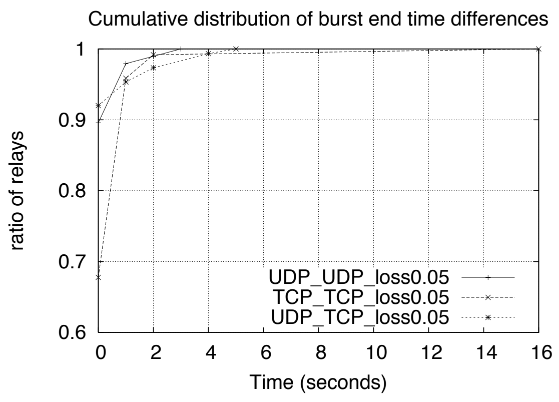

Fig. 8. CDF of burst end time difference

is very close to 1. For the case of 5% packet loss rate, the cumulative distribution for the burst size ratio changes noticeably. In particular, the distribution, which in the case of no artificial packet loss has a single mode, now has more than one mode. Note that in this regime, the TCP_in_TCP_out protocol combination has the largest burst size ratio, as many packets will be retransmitted by the protocol.

B. Experiment II: Traffic relayed over a controlled Skype node

In this second experiment we control and monitor the relay node used by two Skype nodes. By running Skype on host with a powerful CPU (i.e., Pentium 4) and with a good connection to the Internet (e.g., 100Mbps Ethernet) for a prolonged period of time (i.e., 20 days), the host will end up being used as a relay node by other Skype users. Figure 13 illustrates the experimental scenario considered. Note that the relay node is connected to the same hub as the dedicated host used to collect the data. We use tcpdump [15] to capture all packets entering and leaving the relay node. In contrast with the previous experiment, in this experiment the data collection point and the relay node are very close to each other (i.e., in the same LAN).

By running this experiment for 20 days, hundreds of Skype calls were relayed through our controlled host. Our Skype host relayed traffic for Skype hosts located in several parts of the world, including Europe, Asia, and South America. With the data collected in this experiment, we can characterize Skype-relayed traffic of real voice calls from a more vast end-host population.

Although no other application was running on our controlled host, not all traffic entering and leaving the host was Skype voice traffic being relayed. In particular, Skype itself can generate a lot of traffic that is not relayed voice traffic. This occurs if the Skype application running on our host is promoted and starts to execute in "super node" mode, which can perfectly happen in our scenario [13]. Therefore, in order to characterize only relayed voice traffic, we must exclude all "non-voice" traffic from the data collected. Although this is not a trivial task since we cannot be sure if some traffic entering/leaving our host is relayed voice traffic, we are aggressive in our

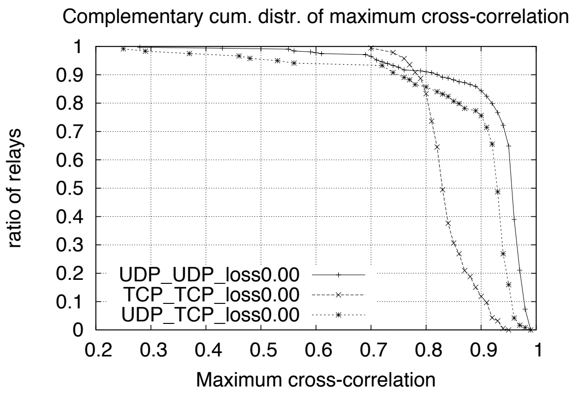

Fig. 9. CCDF of burst maximum cross-correlation

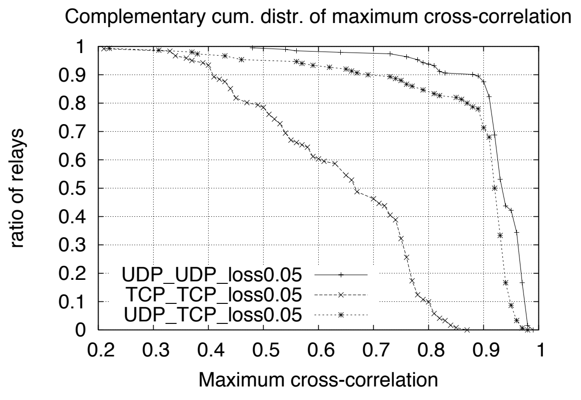

Fig. 10. CCDF of burst maximum cross-correlation

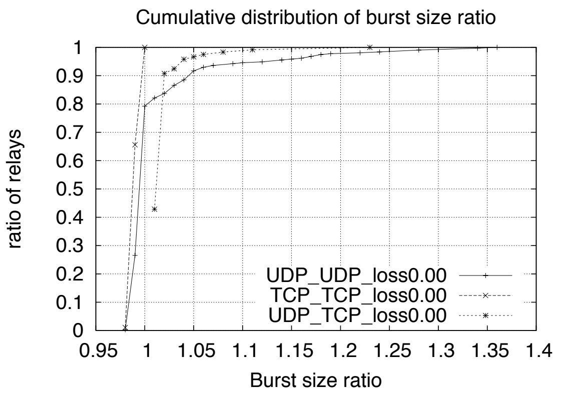

Fig. 11. CDF of burst size ratio

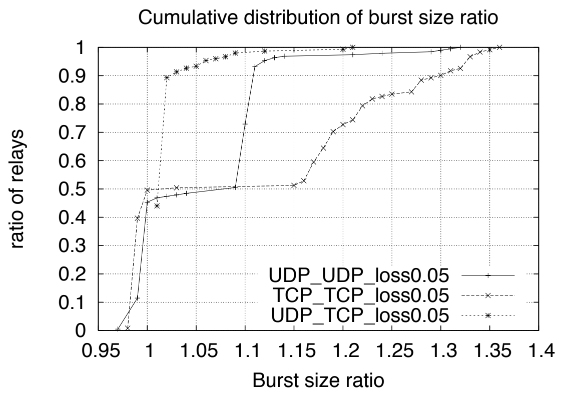

Fig. 12. CDF of burst size ratio

TABLE II
PARAMETERS FOR DETERMINING CANDIDATE RELAYS

Parameter	**value**
Minimum burst duration	30 seconds
Minimum packet count for flow	300 packets
Maximum start time difference	30 seconds
Maximum end time difference	30 seconds
Conflict resolution criteria	Smaller burst start time difference

approach to exclude "non-voice" traffic (i.e., possibly excluding some legitimate relayed voice traffic, but unlikely considering any "non-voice" traffic). Table II shows the parameters used to filter out "non-voice" traffic. Some of these parameters were obtained from the results of the previous experiment. In particular, we use a maximum of 30 seconds for start and end time differences, as such value is much larger than any instance of the previous experiment. Finally, it is possible that more than two bursts of packets start and end within 30 seconds of each other, generating multiple candidates for relayed bursts. In this case, we use a simple prioritizing rule, where the pair of bursts which has the smallest start time difference is chosen as a relay[1].

Over the period of 20 days, we observed a total of 341 Skype-relayed bursts of packets, corresponding to a total of 1.11Gbytes of data. We now characterize the nature of these relays using the same metrics as before. Figures 14 and 15 show the cumulative distribution for the start and the end time differences between two bursts of Skype-relayed traffic, respectively. Note that for 99% of the Skype relayed traffic, the start time difference is less than 6 seconds while the end time difference is less than 10 seconds. Again, we observe that start and end time

[1]However, no conflict needed to be resolved in the data collected for this experiment.

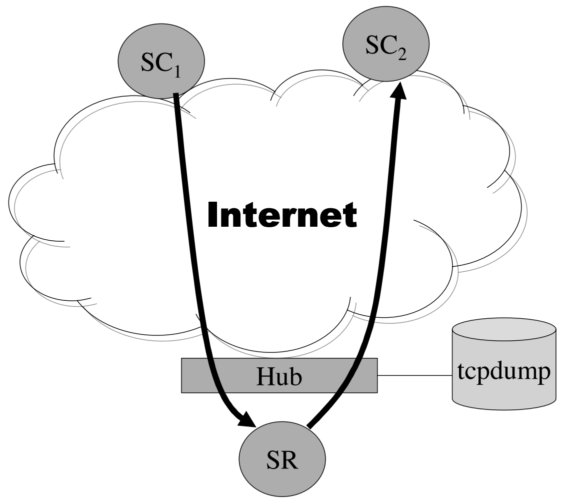

Fig. 13. Configuration of controlled experiment II

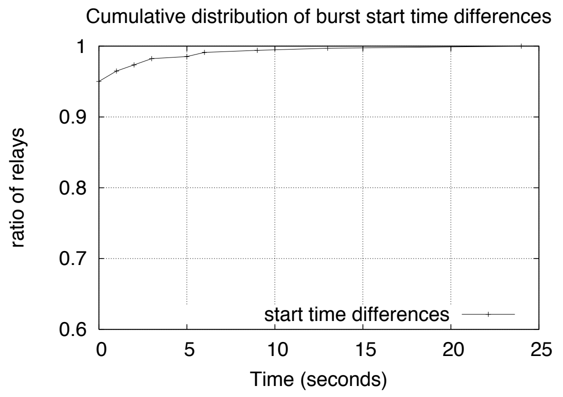

Fig. 14. CDF of burst start time differences of voice calls relayed by our Skype node

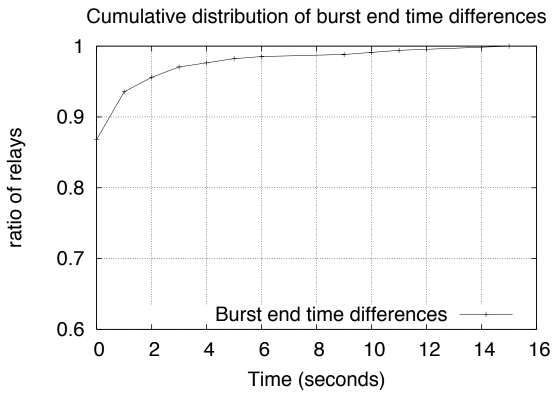

Fig. 15. CDF of burst end time differences of voice calls relayed by our Skype node

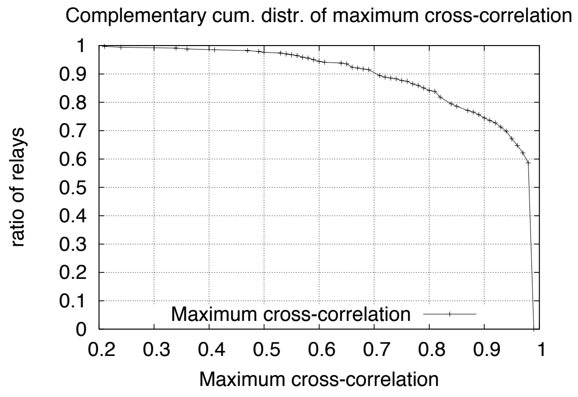

Fig. 16. CCDF of burst maximum cross-correlation of voice calls relayed by our Skype node

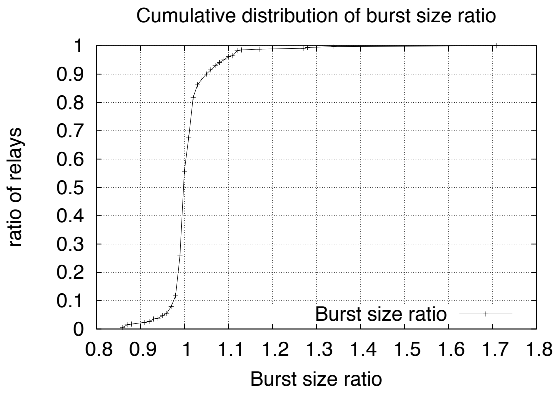

Fig. 17. CDF of burst size ratio of voice calls relayed by our Skype node

differences for the vast majority of Skype-relayed traffic are small. This observation is consistent with the results of the previous experiment when no artificial packet loss is introduced.

Figure 16 shows the complementary cumulative distribution of the maximum cross correlation between two bursts of Skype-relayed traffic. The vast majority of maximum cross correlation observed (i.e., 99% of them) is above 0.41. Note that the distribution for maximum cross correlation is very similar to the distribution of UDP_in_UDP_out and UDP_in_TCP_out protocol pairs of the previous experiment. This is because most of the relays observed in this experiment are of the types UDP_in_UDP_out and UDP_in_TCP_out (i.e., 96% of relays). However, the ratio of relays that have a maximum cross correlation of at least 0.95 is even higher than that of the UDP_in_UDP_out combination in the previous experiment. One possible reason for the higher correlation is that in this experiment, the monitoring point is very close to the relay node (i.e., same LAN). However, in the previous experiment, the monitoring point was very close to the two communicating end-hosts and potentially far from the relay node. This observation is to our advantage as our goal is to detect Skype-relayed traffic by monitoring the access link of a large network, in which case, relay nodes will be close to the monitoring point (e.g., same WAN).

Finally, Figure 17 shows the distribution of the burst size ratio between two bursts corresponding to Skype relayed traffic. Note that 99% of the burst size ratios lie below 1.15, indicating that the vast majority of relayed bursts in Skype have very similar sizes.

IV. Payload-based Skype traffic identification

In this section we describe a heuristic that can be used to identify Skype traffic. Although the heuristic uses Skype-specific information present in the packet payload, our relay detection mechanism does not. The goal here is to filter out non-Skype traffic and obtain a set of relayed-Skype traffic which can then be used as a benchmark for the relay detection mechanism. Note that the heuristic itself may be of interest, as it provides an effective method for identifying Skype traffic when packet payloads are available. However, the heuristic does not detect *Skype-relayed* traffic, it only identifies Skype traffic.

Before describing the heuristic, we first give some details about the functionality of Skype. Along with other new peer-to-peer (P2P) applications, Skype does not use any well-known port number. However, it does use a single port number which is randomly chosen when the application is first installed. This port number is used to receive incoming TCP connections and UDP packets. Moreover, all outgoing UDP packets also have this port number. After it is chosen, the port number is saved locally and used on all future executions of the application. Note however, that the user has the option of explicitly changing this port number[2]. Users of Skype must first login and authenticate themselves before using the application. This login process is composed of two subprocesses: software version verification and user authentication. For the software version verification, Skype makes a TCP connection to a well-known server (i.e., ui.skype.com) and reports the application version currently running [13, 16]. The user authentication is done either by directly contacting the same well-known server or by contacting one or more *super nodes*. Super nodes are instances of Skype applications that have been chosen to perform maintenance work for the application (usually nodes that have a powerful CPU and a fast connection to the Internet). Among other tasks, super nodes provide a distributed directory service for locating Skype users. Every time Skype is executed and during the user login process, the application contacts a few super nodes in order to register itself into the P2P network. This is done by sending UDP packets to the super nodes (to the port they are listening to). The key observation is that these packets have as source port the randomly chosen port number that will be used for Skype traffic [3]. The heuristic works as follows. We first identify the IP address of all hosts that have executed Skype. This is done by inspecting the payload of packets destined to the well-known server. Because of the mandatory version verification process, which is done via a single server, it is easy to determine the IP address of hosts running Skype. However, determining the port number used by a given Skype host to send/receive traffic is more subtle. We correlate the version verification message with the fact that super nodes are contacted in order to obtain the port number used by a given Skype host. Note that these two events usually occur closely to each other (i.e., as soon as the application is launched). In particular, we count how many times a given source port number is used right after a Skype version verification message occurs. If the same port number is used many times to different hosts within the next few packets, we say that this port number is the Skype port.

In order to obtain a higher confidence on the identification of Skype hosts and their respective port numbers, we collected data for 10 days before the packet trace that will soon be analyzed was collected. This allows us to filter out all Skype traffic from this packet trace. However, not all Skype traffic is voice traffic. As discussed earlier, Skype nodes can generate non-voice traffic, such as directory service traffic. We discard such traffic by removing

[2]It is also possible to configure Skype to use TCP port 80 and 443 instead of the randomly chosen port number.

[3]Note that most of the machines in our university network including the machines in dormitory area use fixed IP addresses and are not behind NAT. Also, if a machine is behind a NAT, it does not relay Skype traffic.

TABLE III

DETAILS OF PACKET TRACE COLLECTED AND ANALYZED

Date	**Day**	**Start**	**Dur**	**Direc.**	**Packets**	**Bytes**	**Num. of flows**
2005-05-09	Mon	15:00	17 hours	Bi-direc.	328M	414 GBytes	1501K

flows that do not have any bursts or have a very low average bit rate (i.e., less than 10 kbps). We are left with a packet trace containing only Skype voice traffic.

Finally, we determine the set of Skype relays by evaluating the reduced packet trace. In particular, we compare the respective start and end time of pairs of Skype voice traffic bursts that have a common host within our network and that are in different directions (one flow entering, the other leaving the network). If the respective differences are smaller than 30 seconds, we say that these two bursts are Skype-relayed traffic. This set of relayed traffic will be referred to as the true population of Skype-relayed traffic in the trace collected.

V. DETECTION OF SKYPE-RELAYED TRAFFIC

In this section, we evaluate the effectiveness of using the proposed metrics as a mechanism to detect Skype-relayed traffic. We use the true population of Skype-relayed traffic present in a large packet trace to measure the performance of each metric. After presenting the results, we discuss some learned lessons.

A. Experimental setting

We consider the experimental scenario illustrated in Figure 2, where the access link of a large network is monitored. In particular, we consider our campus network, which is composed of thousands of computers and users. We monitor the Giga-bit access link connecting the campus network to the commercial Internet service provider. A packet capture card (called a *DAG* [17] card) copies all packet headers (including part of the payload) traversing the link (in both directions) to disk along with an accurate time stamp.

Using the monitoring infrastructure, we collect a 17-hour packet trace. Table III shows the details of the trace collected. We filter out Skype voice traffic from this trace and identify the relayed traffic, as explained in Section IV. A total of 381 Skype relays were identified. These relays will provide a benchmark to evaluate our detection mechanism which does not use application-specific information. This set of 381 Skype relays will be referred to as the true Skype relay population.

B. Evaluation of results

In order to evaluate how the metrics proposed fare when detecting Skype-relayed traffic, we will consider true positive and true negative ratios. Formally, the true positive and true negative ratios are given as follows:

- true positive (= 1 - false negative) =

$$\frac{\text{Num. of true Skype relays classified as Skype relays}}{\text{Num. of true Skype relays}}$$

- false positive (= 1 - true negative) =

$$\frac{\text{Num. of false Skype relays classified as Skype relays}}{\text{Num. of false Skype relays}}$$

The entire Skype-like relayed traffic population is obtained by considering as relays all pairs of bursts that conform to the parameters summarized in Table II. This yields a total of 12193 possible relayed bursts, from which only 381 are true Skype relays. Of course, most of these pairs of bursts are not actual relays. By choosing a given threshold for each of the metrics proposed, we can drastically reduce this set, while maintaining most true Skype relays. In particular, each choice of threshold for each parameter will induce a given true positive and true negative ratios.

We will first consider each metric in isolation. This will give insights on how each metric performs, individually, in detecting Skype-relayed traffic. We will also consider setting thresholds for multiple metrics at the same time. Intuitively, this should yield a better detection of Skype-relayed traffic.

Figure 18 shows the true positive and true negative ratios when the start time difference is used as the sole criteria to detect Skype-relayed traffic. Note that each threshold value for start time difference induces a different true positive and true negative ratio. As expected, by increasing the threshold we obtain a higher true positive ratio but at the same time a lower true negative ratio (as more non Skype-relayed traffic is wrongly identified as Skype relays). Note that by considering pairs of bursts that start at most 1 second from each other (threshold of 1 second) we obtain true positive and true negative ratios larger than 0.90. This indicates that the start time difference can effectively be used to correctly identify Skype-relayed traffic. Figure 19 shows the true positive and true negative ratios when the end time difference is used as the sole criteria to detect Skype-relayed traffic. The results are similar to the start time difference.

Figure 20 shows the true positive and true negative ratios when maximum cross correlation is used. Note that this metric alone yields 0.92 true positive ratio and 0.92 true negative ratio (when the threshold is set to 0.55). Within the different metrics, maximum cross correlation provides the best criteria for detection of Skype traffic. Finally, Figure 21 shows the true positive and true negative ratios when the burst size ratio is used.

From these results, we can observe a clear trade-off between true positive and true negative ratios. However, it is desirable to have both ratios as close as possible to one. Note that by using any of the metrics alone it is possible to achieve a 0.88 ratio for both true positive and true negative, under the right choice for the threshold.

We also consider the use of multiple metrics to detect Skype-relayed traffic. In order to obtain the best combination among all possible threshold values for the four metrics, we perform a brute-force search over the parameter space. We discretize the values for each metric (as shown in the figures) and compute the true positive and true negative ratios for each combination of thresholds, i.e., using the thresholds simultaneously. Using this algorithm we can achieve a 0.96 true positive ratio and 0.96 true negative ratio. The thresholds that yields this performance are: 11 seconds for start time difference, 13 seconds for end time difference, 1.33 for burst size ratio, and 0.38 for maximum cross correlation. Interestingly, each of the thresholds corresponds to the threshold that we need to choose in order to obtain 99% of relays in our controlled experiment II respectively [4].

C. Lessons and guidelines

Although maximum cross correlation achieves the best results for a single metric in terms of true positive and true negative ratios, better ratios can be obtained if all metrics are used together. However, such gains are not very large. This occurs because the metrics are somewhat correlated. For example, a relayed traffic burst that has a large start time and end time difference is likely to have a low maximum cross correlation.

We have also noticed that traffic generated by applications other than Skype is sometimes misclassified as being Skype-relayed traffic. By visually inspecting the time series associated with the bursts and looking at port numbers used, we identified several of these applications. A few examples are a popular online arcade game (i.e., Kartrider), a Gnutella-variant P2P file sharing application (i.e., Bearshare), and a popular online game running on XBOX. Interestingly, it is possible that some of the traffic identified as Skype-relayed traffic is indeed relayed traffic (albeit not Skype). In particular, some of these false positives have a very high maximum cross-correlation and a burst size ratio near one. We suspect that these may indeed be other types of relays. After further investigation, we found the following:

- TCP traffic generated by Bearshare
  Queries and query responses transmitted over TCP and UDP have been adopted by developers and software vendors of Gnutella-based file sharing applications [18]. More specifically, queries search for content and search results can be relayed through hosts running the Gnutella application. These clients usually use the

[4]See Figures 14, 15, 16, and 17.

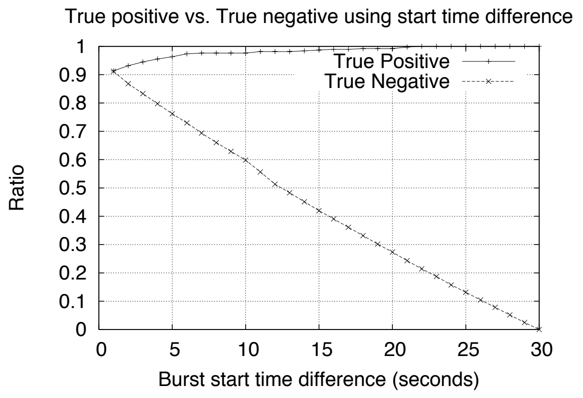

Fig. 18. True positive vs. True negative using start time difference as a classifier

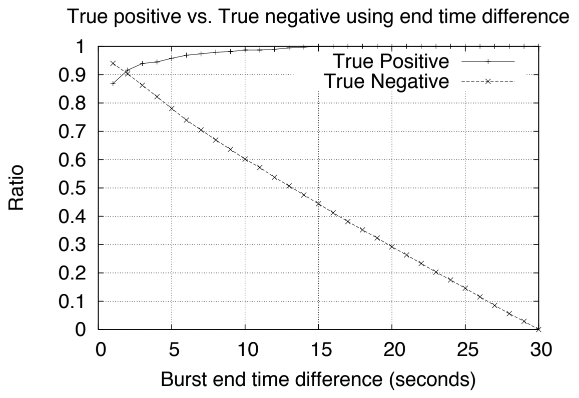

Fig. 19. True positive vs. True negative using end time difference as a classifier

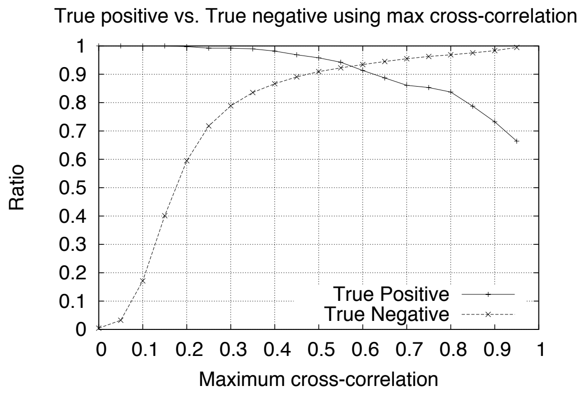

Fig. 20. True positive vs. True negative using maximum cross correlation as a classifier

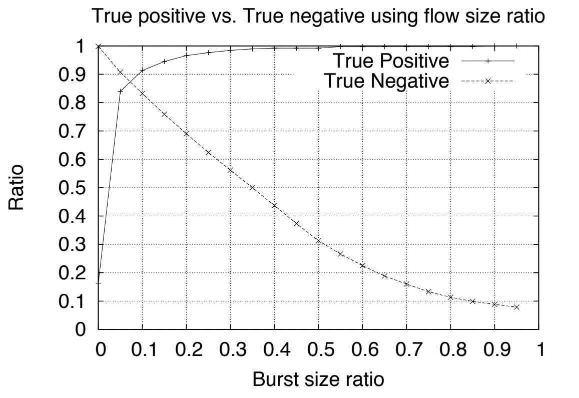

Fig. 21. True positive vs. True negative using burst size ratio as a classifier

default port number (i.e., 6346). We identified several Gnutella clients in our packet trace that seem to have relayed traffic.

- UDP traffic generated by P2P online game and XBOX Live
  We identified several P2P online games which generate traffic that could be classified as relayed traffic. The pair of bursts generated by these applications have a relatively high maximum cross correlation or burst size ratios close to one, and very small start and end time differences. These applications were Kartrider online game and some other XBOX online game. At this moment, we cannot be sure if these applications indeed relay traffic over end-hosts. Further analysis is needed, as these protocols are proprietary and it is not trivial to understand exactly what is happening. We note, however, that most of these supposedly relayed traffic do not have a high maximum cross-correlation and a burst size ratio close to one at the same time. Also, for the most of the cases, the number of bursts that are detected as carrying relayed traffic by our technique in each flow is fairly small compared to the total number of bursts in the flow. This tends to indicate that such bursts may not be traffic relays.

We explicitly note here that a relay detection mechanism based on the metrics proposed can be easily extended to other applications, such as End System Multicast (ESM) [4] and SQUID proxy cache [19]. We conducted some preliminary controlled experiments using these two applications and found that most of the metrics used are directly applicable in these cases. However, some metrics may have to be slightly modified, specially for streaming application such as ESM. Unlike Skype where relay nodes are not interested in the data passing through them, relay nodes in ESM are potentially interested in the data (i.e., they are also consuming the data). The implication is that the start time difference and the end time difference for ESM relayed traffic need not be necessarily small. In fact, they can be arbitrarily large. Figure 22 illustrates a large start time difference between two bursts of ESM-relayed traffic. This occurs because the first client starts receiving the multimedia stream much before the second client joins the multicast group. Note that the first client only starts relaying traffic after the second client joins. Instead of using start and end time differences, one possibility might be to introduce the concept of burst "containment", where a given burst is "contained" in another. Finally, SQUID proxy cache can also generate relayed traffic which can be detected by our technique. In particular, if requested objects are large enough (e.g., hundreds of Kbytes), this will generate large bursts of packets that can be identified and then matched. One caviat is the effect of caching done by SQUID. However, since we use bursts of packets (as opposed to flows), this should not introduce any serious problems.

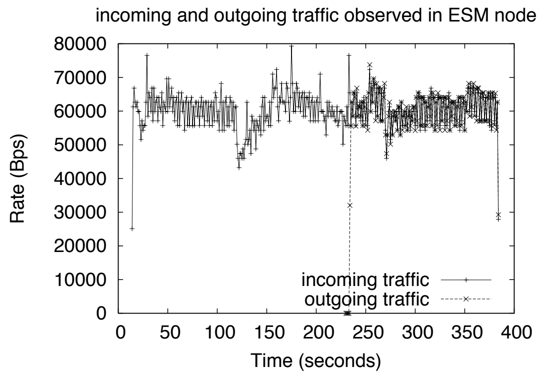

Fig. 22. Example of relaying traffic in ESM

VI. Related work

Detecting relayed traffic by monitoring an access link is a topic that has received some recent attention, particularly within the context of intrusion detection [7–9]. However, most of the efforts have focused on detecting relayed traffic generated by interactive applications. In particular, timing-based techniques have been proposed to detect hosts used as relays in a connection chain of sequential logins (known as stepping stones) [7, 8]. These techniques rely on the ON-OFF behavior of interactive applications and require connections to have a bounded OFF period. Such techniques do not apply to multimedia applications, where traffic patterns do not exhibit an ON-OFF behavior.

More recently Sekar et al. [9] suggested using the temporal order of flow start times to identify nodes that are being used as relays. The use of temporal order of start times is similar to our idea of using start time differences. However, their work is preliminary and does not characterize or evaluate the effectiveness of detecting relays based on temporal ordering of flows. Moreover, the use of temporal ordering of flows alone will not lead to an effective criteria to detect Skype-relayed traffic (as the false positive ratio will be very high).

Another related area of work is the classification of network traffic based on the application generating the data. Several methods to classify traffic have been proposed in the literature [6, 20–23]. Traffic classification can help the detection of relayed traffic when the applications that use relay nodes are known and can be identified. For example, traffic classification can be performed by profiling application-specific signatures [20, 23]. However, we focus on applications that cannot be easily profiled and propose a technique that does not rely on any application- or protocol-specific information.

Finally, Hussain et al. [24] have proposed a methodology to identify source nodes of Denial-of-Service (DoS) attacks using frequency spectrum analysis of traffic generated. Their approach assumes that the traffic generated by two successive attacks by a given host connected to a given ISP exhibit similar frequency spectrum. Unfortunately, a similar idea would not work well for the detecion of relay nodes. The main reason is that a given relay node can be used by several different pairs of hosts, each pair having its own unique traffic signature.

VII. Conclusion

In this paper, we have characterized the nature of Skype-relayed traffic using different metrics. In particular, we have proposed the following metrics: start and end time differences, byte size ratio, and maximum cross correlation between two relayed *bursts of packets*. Using these metrics, we propose a methodology for detection of Skype-relayed traffic based on thresholds. Our approach relies solely on flow-level traffic characteristics, rather than on application- or protocol-specific information.

In order to generate and collect a large amount of relayed traffic, we have created two different controlled experimental environments using Skype. In the first environment, voice traffic generated by two nodes under our control is relayed over some node in the Internet. In the second environment, two Skype nodes use a relay node under our control to communicate. The data collected was characterized using the metrics we proposed.

We apply our detection methodology to an aggregate packet trace collected at the access point of our university. Our results show that the proposed metrics can be reliably used to detect Skype-relayed traffic, yielding both low false positive and low false negative ratios. Particularly, when using the maximum cross correlation as the sole criteria for detecting Skype-relayed traffic we obtain a 8% false negative and 8% false positive ratios. However, we demonstrate that by simultaneously using multiple metrics we can achieve an even higher accuracy. Particularly, we obtain a 4% false negative and 4% false positive ratios when all metrics are used. Because of such improvements, we argue that multiple criteria should be used when detecting relayed traffic.

As part of on-going work, we are currently evaluating our methodology on other multimedia applications, such as End System Multicast, and considering it to detect relayed traffic generated by applications that can have a high data rate, such as SQUID proxy cache. In addition, we plan to extend our methodology to consider relay nodes that perform more complex transformations, such as compression/decompression and CODEC changes.

ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation under NSF grants ANI-0085848, ANI-0240487, ANI-0085848, EIA-0131886, and EIA-0080119. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies. We also would like to thank Tyler Trafford at UMass for his support and assistance in setting up the controlled experiment environment.

REFERENCES

[1] B. Ford, P. Srisuresh, and D. Kegel, “Peer-to-peer communication across network address translators,” in *Proc. of USENIX annual Technical conference*, April 2005.

[2] N. Feamster, D. Andersen, H. Balakrishnan, and F. Kaashoek, “Measuring the effects of Internet path faults on reactive routing,” in *Proc. ACM SIGMETRICS*, June 2003.

[3] Skype,
`http://www.skype.com`.

[4] Y. hu, S. Rao, S. Seshan, and H. Zhang, “Enabling conferencing applications on the Internet using an overlay multicast architecture,” in *Proc. ACM SIGCOMM*, August 2001,
`http://esm.cs.cmu.edu`.

[5] J. Oikarinen and D. Reed, “Internet relay chat protocol RFC,” 1993.

[6] T. Karagiannis, K. Papagiannaki, and M. faloutsos, “BLINC: Multilevel traffic classification in the dark,” in *Proc. ACM SIGCOMM*, August 2005.

[7] Y. Zhang and V. Paxson, “Detecting stepping stones,” in *Proc. of USENIX security symposium*, August 2000.

[8] D. Donoho, A. Flesia, U. Shankar, V. Paxon, J. Coit, and S. Staniford, “Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay,” in *Proc. of Int’l Symp. on Recent advances in intrusion detection*, 2002.

[9] V. Sekar, Y. Xie, D. Maltz, M. Reiter, and H. Zhang, “Toward a framework for Internet forensic analysis,” in *ACM workshop on Hot Topics in Networks (HotNets)*, November 2004.

[10] Cisco Netflow,
`www.cisco.com/warp/public/732/Tech/netflow`.

[11] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, “On the characteristics and origins of Internet flow rates,” in *Proc. ACM SIGCOMM*, August 2002.

[12] P. Bourke, “Cross correlation,” August 1996,
`http://astronomy.swin.edu.au/~pbourke/ analysis/correlate`.

[13] S. Baset and H. Schulzrinne, “An analysis of the skype peer-to-peer internet telephony protocol,” Columbia university, Tech. Rep. TR-CUCS-039-04, 2004.

[14] Dummynet,
`http://www.dummynet.com`.

[15] Tcpdump,
`http://www.tcpdump.org`.

[16] F. Bulk, “Final project:skype,” May 2004.

[17] Endace, “Endace network monitoring cards,” 2005,
`http://www.endace.com`.

[18] Gnutella–Fact, info and encyclopedia article,
`http://www.absoluteastronomy.com/ encyclopedia/G/Gn/Gnutella.htm`.

[19] Squid Web Proxy cache,
`http://www.squid-cache.org`.

[20] A. Moore and D. Zuev, “Internet traffic classification using bayesian analysis techniques,” in *Proc. ACM SIGMETRICS*, June 2005.

[21] A. Moore and K. Papagiannaki, “Toward the accurate identification of network applications,” in *Proc. of Passive and Active measurement workshop*, April 2005.

[22] C. Dewes, A. Wichmann, and A. Feldmann, “An analysis of internet chat systems,” in *Proc. of ACM Internet Measurement Conference (IMC)*, October 2003.

[23] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, “Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification,” in *Proc. of ACM Internet Measurement Conference (IMC)*, October 2004.

[24] A. Hussain, J. Heidemann, and C. Papadopoulos, “Identification of repeated attacks using network traffic forensics,” USC/Information Sciences Institute, Tech. Rep. ISI-TR-2003-569b, 2003.