Identifying Skype Nodes by NetFlow-based Graph Analysis

Jan Jusko and Martin Rehak
Faculty of Electrical Engineering, Czech Technical University in Prague
Cognitive Security s.r.o., Prague, Czech Republic
{jan.jusko, martin.rehak}@fel.cvut.cz

Abstract—**In this paper we present an algorithm that is able to progressively discover nodes of a Skype overlay P2P network. Starting from a single, known Skype node, we can easily identify other Skype nodes in the network, through the analysis of widely available and standardized IPFIX (NetFlow) data. Instead of relying on the analysis of content characteristics or packet properties, we monitor connections of known Skype nodes in the network and then progressively discover other nodes through the analysis of their mutual contacts. We show that our results are comparable to the methods using more complex data analytics. The use of standardized input data allows for easy deployment onto real networks. Moreover, because this approach requires only short processing times, it scales very well in larger and higher speed networks.**

I. Introduction

As recently shown in [10], peer-to-peer (P2P) traffic not only consumes a significant amount of bandwidth, but it can also degrade the performance of anomaly detection techniques. The detection rate can decrease by up to 30% and false positive rate can increase by up to 45%. Even though the study has focused primarily on file sharing P2P traffic, it states that VoIP telephony has similar effects on anomaly detection performance. In this paper we deal specifically with Skype, which is a VoIP application based on a P2P protocol.

Skype appears to be a perfect example of VoIP-telephony popularity these days. It allows for easy use of telephony, instant chat, file transfer, and other services. In contrast to other VoIP applications, Skype is able to connect to its network even behind restrictive firewalls or NATs, and its client application is available across several platforms, including mobile phones. In March 2011 Skype recorded more than 30 million users online [11]. Overall user count was reported to be 663 Million [4].

There is considerable demand for the detection of Skype's traffic. Some ISPs claim that it congests their networks and by blocking it, they can improve their Quality of Service. Also, some 3G operators want to block Skype usage, since they believe it cuts into their profits. And as has been already mentioned, correct detection of Skype's traffic can improve the performance of anomaly detectors, and consequently lead to better security.

In this paper we focus on detecting Skype nodes in the network using only NetFlow data. Our approach is rather different from those presented so far: instead of packet inspection and flow analysis we focus on reconstructing the Skype overlay network via a single Skype node, further called the *seed*.

II. Related Work

It is believed that Skype is related to KaZaA [9], which is a client application built on the FastTrack P2P protocol. It is not known whether Skype uses FastTrack directly or some similar protocol, but FastTrack and Skype have several similarities that suggest that they are closely related. For example, they both use unstructured two-layer P2P network with ordinary nodes in one layer and "super-nodes" in other layer.

A basic description of Skype architecture and an analysis of the message workflow can be found in [8]. Authors have also created several signatures to detect Skype traffic based on their findings. Detection using signatures requires packet payload inspection, which is processor intensive and introduces significant latency. Thus is not suitable for high-bandwidth networks. Another analysis of Skype can be found in [9]. In this work the authors also provide an analysis of user-behavior and estimate network workload generated by Skype.

In the field of Skype detection we can find several approaches. In one of the first [6], the authors devised two detection methods — one based on Pearson's Chi Square test and the other using Naive Bayes Classifiers. Both methods require packet payload inspection and therefore their deployment on backbone networks can be problematic. These two methods were further modified and their performance analyzed in [13]. Packet inspection and flow properties are also used for detection in [17]. Another approach, introduced in [16], is based solely on the flow data from the network.

An interesting Skype detection method designed specifically for 3G networks can be found in [14]. The authors exploit a specific property of 3G networks whereby every packet is associated with a specific mobile device (and thus user). However, it is still necessary to inspect packet payloads in order to detect active Skype node in the network.

Skype is analyzed from a rather different perspective in [15] where two Skype outages are analyzed and compared. One of the interesting points of this work is that Skype temporarily centralized its topology by the introduction of so-called *mega-supernodes*, thus speeding up the network recovery.

An approach similar to ours was used in [7] to detect members of a P2P botnet. Their detection also starts with one known node of a given P2P network and was based on

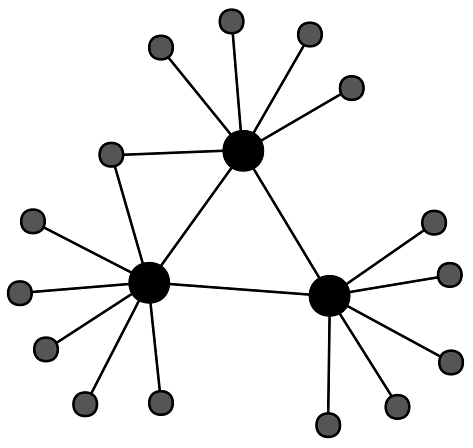


Fig. 1. Visualization of a Skype network. Larger black nodes represent supernodes, smaller gray nodes represent clients. Links between nodes represent connections

monitoring of mutual contacts. However, they used a rather different graph representation and then determine the detected node's confidence after the graph is constructed.

III. Detection Method

P2P networks can be represented by a graph, in which vertices represent nodes participating in the P2P network and edges represent connections between two nodes participating in the P2P network. An example of such a structure (in this case Skype) can be found in Fig. 1. We try to exploit this graph structure to find other participating P2P nodes network using one starter node. To achieve this, we traverse the edges of the graph which is constructed on the basis of observed network communication. Since P2P overlay networks are dynamically changing, so should the graph that represents a P2P overlay network.

To detect the nodes of a P2P overlay network within our network we use a 3-partite weighted graph

$$G = (V, E, w)$$

where

$$V = V_c \cup V_s \cup V_r.$$

V_c is a set of nodes from our network we *believe* are participating in the P2P network, V_s is a set of nodes from our network that we *suspect* are participating in the P2P network and V_r is a set of nodes from outside of our network communicating with nodes from $V_c \cup V_s$. E is a set of edges. Function w assigns each edge a weight — a value equal to the time when the edge was added to the graph. We ignore all intra-network communication and cannot see communication between the nodes that are outside of our network. Therefore the graph we define is indeed a 3-partite weighted graph. This also implies that $G = ((V_c \cup V_s) \cup V_r, E)$ can be considered a bipartite graph.

The detection algorithm monitors network traffic and constructs (modifies) the graph based on the observed network activity in the following way:

- the graph starts with only the seed node $n \in V_c$,
- when a network connection occurs between any node $n \in V_c$ and some node m outside of our network then there are two options:
  - $m \in V_r$ already; in this case we just update $w(\{m, n\}) = current\_time()$,
  - $m \notin V_r$ yet; in this case we add m to V_r and $\{m, n\}$ to E and set $w(\{m, n\}) = current\_time()$.
- when a network connection occurs between any local node not yet in the graph and some node $m \in V_r$ we add n to V_s, add $\{m, n\}$ to E and set $w(\{m, n\}) = current\_time()$,
- any edge $e \in E$ for which $t_{now} - w(e) > t_L$ is removed from the graph,
- any node $n \in V$ is removed from the graph when it does not have any incident edge (it has a zero degree),
- if $(\exists m \in V_s)(\exists n \in V_c)(\mid Adj(m) \cap Adj(n) \mid > K)$ then we move m from V_s to V_c, where $Adj(n)$ is a set of vertices adjacent to n.

The output of the algorithm is the set V_c which at any given moment contains a list of active P2P nodes in the local network. There are two parameters used in this algorithm:

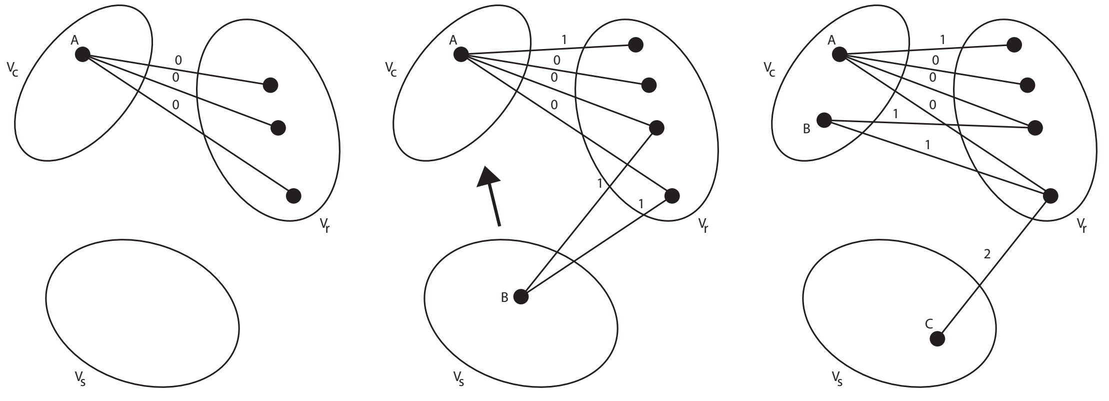
- a *memory limit*, t_L, which specifies how long a recorded connection (an edge in the graph) is kept in memory,
- a *mutual contacts overlap threshold*, K, which specifies how many mutual adjacent vertices a node from V_s needs to have with any node from V_c to be moved to V_c, i.e. to consider it a P2P node.

A. Using the P2P Detection Mechanism to Detect Skype

Besides being a VoIP application, Skype is also a P2P application that needs to dedicate a portion of its communication to maintain a functional overlay network.

Scenarios where a Skype node communicates with other nodes in the network have been already documented. For example, every time a Skype client connects to the Skype network (when a user signs in), it contacts up to 22 distinct nodes in the Skype overlay P2P network [3]. Searching a user in the Skype network also induces several connections within the overlay network. In both cases the connection is made over UDP protocol, unless the node is behind a UDP restricted firewall, in which case TCP is used. In addition to these observations, on average Skype nodes makes 16 connections to other nodes within each 5 minute interval, with number of distinct contacted nodes rising linearly with time [12].

Based on the knowledge gathered so far about Skype and its protocol we are able to adjust the generic algorithm for the purpose of Skype detection. We know that Skype uses a single principal port to listen for both incoming TCP and UDP connections. This port is chosen when the application is launched for the first time and never changes, unless a user does so manually. Based on this knowledge we use tuple (ip, port) as a node in the graph used by the detection algorithm. We call this tuple an *endpoint*. Since this port is always chosen from ports higher than 1024 we can ignore all traffic on ports



(a) 1. time interval (b) 2. time interval (c) 3.time interval

Fig. 2. Algorithm illustration. First we have a seed node A with 3 recorded contacts. In the second time interval, another node, B, is observed, sharing two mutual contacts with A. If we consider $K = 2$, then in the third step, node N is already moved to the V_c. Moreover, the algorithm detected yet another node, which has only one mutual contact with a node from V_c. Note that the weights of the edges in the graph are determined by the time step in which they occured most recently.

lower than that, effectively ignoring most of the web traffic. Skype supernodes can and do use ports 80 and 443 to listen for incoming TCP connections, but these ports are always used together with the main port which is always higher than 1024. We believe that our generic algorithm can also be adjusted to be used with other P2P networks.

IV. Algorithm Aplicability for Skype

We need to verify that our proposed algorithm is able to detect Skype nodes in the network based on the knowledge of a single node. We used one of the datasets provided at [1] for this purpose. These standard datasets were also utilized in [6], [2], [5], [13]. In particular, we use the dataset containing *only* Skype signaling traffic flowing to and from Skype's main ports (not ports 80 and 443) over UDP that was identified by the method proposed in [6] and spans over 4 days.

We process this data set in five-minute batches. Every 5 minute interval represents one unit of time in the sense of evaluation $w(e), e \in E$, i.e. edge e_1 induced by a connection in the first time interval would have $w(e_1) = 0$, edge e_2 induced by a connection in the second time interval would have $w(e_2) = 1$ and so on. We also define several new terms — *mutual contacts overlap* for two nodes m, n and memory limit t_L, *average maximum overlap* of node n, $\bar{O}^t(n)$, and *upper detection bound* for mutual contacts overlap threshold k, $\bar{U}(k)$:

$$\begin{aligned} o^t(m,n) &= \mid Adj(m) \cap Adj(n) \mid \\ \bar{O}^t(n) &= \frac{\sum\limits_{t_i \in T_{active}} \max\limits_{m \in N \setminus \{n\}} o^t_{t_i}(m,n)}{\mid T_{active} \mid} \\ \bar{U}(k,t) &= 100 \frac{\mid \{n \in N \mid \bar{O}^t(n) \geq l\} \mid}{\mid N \mid}, \end{aligned}$$

where $o^t(m,n)$ is calculated on a graph with $t_L = t$, T_{active} is a set of time intervals in which node n was active, N is set of all nodes in the data set, and $o^t_{t_i}(m,n)$ is $o^t(m,n)$ measured in time interval t_i. The term $\bar{O}(n)$ represents the node's average of maximal overlaps across all time windows in which the node was active, thus it can be considered as a measure of its detectability by our method. $\bar{U}(k,t)$ shows what percentage of Skype nodes in our network could be detected by our algorithm if all other nodes in the network are already known, using $K = k$ and $t_L = t$.

To determine the viability of the proposed algorithm we ran the detection algorithm for several combinations of K and t_L. At each time step we determined $o^{t_L}(m,n) \forall n \in N, \forall m \in N - \{n\}$ and subsequently stored the calculated value. After each run we calculated $\bar{O}^{t_L}(n) \forall n \in N$ and $\bar{U}(k, t_L)$, whose calculated values can be found in Fig.3. The higher the $\bar{U}(k, t_L)$ gets, the better the detection performance we can expect. The graph shows that for each choice of K there is a choice of t_L after which the $\bar{U}(k, t_L)$ does not yield any significant improvement

Based on these results we believe that the algorithm is capable of efficiently tracking Skype nodes in the network.

V. Evaluation

In the experiments we used data sets described in Section V-A with values of $K \in \hat{7}$ and $t_L \in \{5, 15, 30, 60, 90, 120\}$. For every combination of these two parameters we ran the algorithm 18 times, using different seed nodes from the control set. Based on these experiments we measured the detection rate, false positive rate, detection speed, processing speed and the impact of the seed node choice.

A. Datasets Description

We perform experiments on a part of our University network, which contains approximately 950 unique IP addresses and many more hosts, some of which are hidden behind a gateway. Due to established network policies we are not

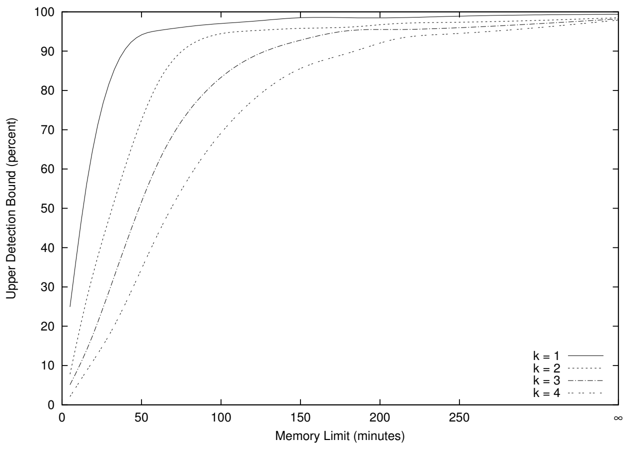


Fig. 3. Obtained upper detection bound $\bar{U}$ for $K \in 1, 2, 3, 4$ and $t \in 5, 15, 30, 60, 90, 120, 150, 180, 210, 240, \infty$, ∞ meaning we never removed any edge from the graph. We can see that for each choice of K there is a choice of t_L after which the $\bar{U}(K, t_L)$ does not yield any significant improvement. B-spline curves are used to create the smooth curve from the observed values.

able to determine the ground truth for all endpoints in the network, therefore we used a small *control set*, comprising of 24 hosts under our control, 18 of which are running Skype. The hosts from the control set are running either Windows or Linux operating systems. We used seven different versions of Skype clients, the oldest one being 4.2.0.87. Any Skype nodes detected by our algorithm was scanned by `nmap` or verified manually to determine whether it is a true or false positive. The set was collected on a 1 Gbit network for 24 hours. We refer to this data as *Host Set 1*.

For the time performance analysis we used yet another data set. It was collected over 24 hours on a 40 Gbit network belonging to a privately held company. Since we have no means of establishing the ground truth for this data set or even using `nmap` to probe the detected nodes, we use this data set solely for the purpose of time performance evaluation. We refer to this data as *Host Set 2*.

B. Detection Rate

Since we did not know the ground truth for the whole data set, we only determined the detection rate on the control set. The detection rate is calculated as an average over the 18 runs. The results can be found in Table I. We have several parameters to choose from that reached a 100% detection rate. The data implies that raising the mutual contacts threshold while keeping the memory limit lowers the detection rate. On the other hand, raising the memory limit while keeping the mutual contacts threshold fixed improves the detection rate.

C. False Positive Rate

We recognize two types of false positive rates:

FPR1 is calculated on the set of *endpoints* encountered during the run of the detection algorithm,

FPR2 is determined on the set of *flows* encountered during the run of the detection algorithm.

TABLE I
ATTAINED DETECTION RATE IN % ONTHE CONTROL SET FROM THE HOST SET 1.

K \ t_L	5	15	30	60	90	120
1	100	100	100	100	100	100
2	96.605	100	100	100	100	100
3	93.519	100	100	100	100	100
4	93.21	100	100	100	100	100
5	73.457	96.605	100	100	100	100
6	65.432	84.877	98.457	100	100	100
7	55.556	80.247	90.123	100	100	100

TABLE II
OBSERVED FPR1 IN % ON HOST SET 1.

K \ t_L	5	15	30	60	90	120
1	0.006	0.006	0.006	0.007	0.007	0.007
2	0.001	0.002	0.003	0.003	0.003	0.003
3	0.002	0.002	0.002	0.002	0.003	0.003
4	0.002	0.002	0.002	0.002	0.002	0.002
5	0.001	0.002	0.002	0.002	0.002	0.002
6	0.001	0.002	0.002	0.002	0.002	0.002
7	0.001	0.002	0.002	0.002	0.002	0.002

The algorithm managed to keep a zero false positive rate on the control set with all parameters we experimented with, thus $FPR1 = FPR2 = 0$. This might be caused by the small set of controlled hosts not running Skype and/or a limited scope of Internet activity on these hosts. We do not expect the algorithm to a keep zero false positive rate on very large networks, where a lot of anomalous behavior would be observed.

We estimated the false positive rates on the Host Set 1 with an assumption of a 100% detection rate (since we were not able to determine the exact detection rate). Again, the final false positive rate for each combination of algorithm parameters was determined as an average over the 18 runs. Calculated FPR1 and FPR2 can be found in Table II and Table III respectively. You can see that while the false positive rate is no longer zero, it is still very low. We were further interested what do these false positives represent. After some investigation, we were able to divide these false positives into three categories:

- gateways (52%),
- suspected former Skype nodes (37%),
- endpoints used by Skype but not associated with the main port (11%).

False positives from the first group were the most common and represent endpoints created by a router or host that serves as a gateway for a network behind a NAT. These endpoints are in fact created by Skype, but the actual host on which the application runs is hidden from the detection engine. Endpoints belonging to gateways once used by Skype can be later reused by another application, thus they are considered false positives. We can avoid these using a very simple gateway detection mechanism whereby any host that is represented by at least 3 Skype nodes is considered a gateway, thus removed from the model and ignored in further processing.

TABLE III
OBSERVED FPR2 IN % ON HOST SET 1.

K \ t_L	5	15	30	60	90	120
1	0.529	0.508	0.508	0.511	0.511	0.511
2	0.359	0.413	0.424	0.401	0.401	0.401
3	0.391	0.376	0.421	0.383	0.386	0.397
4	0.394	0.376	0.376	0.385	0.385	0.385
5	0.399	0.456	0.456	0.459	0.459	0.462
6	0.407	0.456	0.456	0.458	0.468	0.468
7	0.405	0.481	0.389	0.389	0.391	0.391

Former Skype nodes are endpoints used by Skype which are no longer utilized — for example when client is shut down. These endpoints are targets of incoming UDP connections from Skype nodes that are yet not aware of this change. We can eliminate these false positives by filtering out all detected nodes that do not initialize an outgoing connection.

The Skype application uses typically one main port and supernodes also ports 80 and 443. Besides that, Skype also opens several ephemeral ports for outgoing TCP connections. We postulate that these ephemeral ports are used for relaying other nodes' traffic. Despite ignoring outgoing TCP connections, these are endpoints associated with ephemeral ports created by incoming connections. Such endpoints comprise the third group listed above. These are not considered typical false positives but it is difficult to establish the ground truth for them. As a precaution, we consider them to be false positives. We do not see any effective way of filtering them, without limiting processed flows to the UDP protocol which would prevent us from detecting nodes behind UDP restricted firewalls.

D. Detection Speed

The detection method proposed in this paper does not classify flows, but rather tries to determine which endpoints belong to the Skype overlay network. This limits our ability to classify a flow immediately when it is first encountered. Our method needs some time to build the graphs and based on that graph we can determine which flows are induced by Skype — based on the endpoints sending or receiving the flow. Our algorithm processes flows in 5 minute batches therefore we can measure the detection speed via how many batches have to be processed for a new Skype instance to be detected. According to our observations of 18 hosts with Skype client from the control set, 13 were detected within the first processed batch, 4 were detected within two batches and the remaining single node was detected after one hour (12 batches). This shows that while the detection is not immediate, the algorithm is able to detect new Skype instances within couple of minutes, in a majority of cases.

E. Algorithm Time Performance

We evaluated the time performance of the algorithm on the Host Set 2 for $K = 2$ and $t_L = 24$ (120 minutes). First we manually found a host running Skype in the data set by verifying that it connects to Skype bootstrap nodes, and used it as a seed to our algorithm. Using this seed node we ran the algorithm three times and measured how much it took to process the traffic over a 24 hour period. Our results show that the algorithm ran for 755, 757 and 772 seconds — less than 3 seconds to process a 5 minute batch of traffic on average.

Furthermore, we measured how much memory the algorithm consumes. We determined that the algorithm itself consumed less than 500 MB of memory.

TABLE IV
DETECTION RATE OF OUR ALGORITHM ON *Host Set 1* IN % USING INTERSECTION OF PARTIAL RESULTS AS A FINAL RESULT.

K \ t_L	5	15	30	60	90	120
1	100	100	100	100	100	100
2	94.444	100	100	100	100	100
3	88.889	100	100	100	100	100
4	88.889	100	100	100	100	100
5	5.556	94.444	100	100	100	100
6	0	61.111	94.444	100	100	100
7	0	61.111	77.778	100	100	100

TABLE V
PERCENTAGE OF DETECTED NODES (BOTH TRUE AND FALSE POSITIVES) PRESENT IN THE INTERSECTION OF INDIVIDUAL DETECTION RESULTS ON HOST SET 1.

K \ t_L	5	15	30	60	90	120
1	87.218	89.096	89.306	89.508	89.508	89.508
2	84.899	85.916	87.972	88.684	88.684	88.684
3	81.463	85.013	86.282	84.838	84.709	85.735
4	79.214	84.687	84.687	86.86	86.415	86.282
5	5.097	86.369	87.843	87.166	89.167	89.038
6	0	69.511	85.462	87.696	86.784	88.847
7	0	66.924	73.786	82.006	81.076	83.654

F. Impact of the Choice of the Seed Node on the Detection Results

While we stated above that the algorithm has two parameters, there is also a third, implicit parameter — the seed node. The algorithm do not necessarily provide the same results for any choice of the seed node. So for each combination of parameters we ran the algorithm 18 times using different seed nodes from the control set and determined the intersection of the individual results.

Table IV shows the obtained detection rate on this intersection. While the seed node choice does impact the detection rate, the algorithm is still capable of attaining a 100% detection rate. Percentage of all detected nodes (both true and false positives) present in the intersection of the partial results can be found in Table V. This implies that using the intersection of partial results usually filters out the false positives.

As expected, the choice of the seed node also impacts the false positive rate. The estimated values of FPR1 and FPR2 (for a 100% detection rate) can be found in Table VI and Table VII respectively. The observed false positive rate is lower than for a single seed node scenario. Most of the eliminated false positives were gateways.

Since we are able to keep a 100% detection rate on the control set and lower the false positive rate by using the

TABLE VI

FPR1 in % for *Hosts Set 1* when using intersection of partial detection results as a final detection result. We can see that the false positive rate is lower compared to one attained using only one instance of the detection algorithm.

K \ t_L	5	15	30	60	90	120
1	0.004	0.005	0.005	0.005	0.005	0.005
2	0	0.001	0.001	0.001	0.001	0.001
3	0	0	0	0.001	0.001	0.001
4	0	0	0	0.001	0.001	0.001
5	0	0.001	0.001	0.001	0.001	0.001
6	0	0.001	0.001	0.001	0.001	0.001
7	0	0	0	0	0	0

TABLE VII

FPR2 in % for *Hosts Set 1* when using intersection of partial detection results as a final detection result. We can see that the false positive rate is lower compared to one attained using only one instance of the detection algorithm.

K \ t_L	5	15	30	60	90	120
1	0.18	0.214	0.214	0.217	0.217	0.217
2	0.102	0.108	0.125	0.144	0.144	0.144
3	0.102	0.102	0.108	0.108	0.108	0.125
4	0.102	0.102	0.102	0.108	0.108	0.108
5	0	0.35	0.35	0.356	0.356	0.356
6	0	0.35	0.35	0.35	0.35	0.35
7	0	0.102	0.102	0.102	0.102	0.102

intersection of results, this is a viable option for improving the detector performance. However, we must also consider the performance issues. According to the observations presented in Section V-E, it is feasible to use several parallel instances of the detector. This parallel execution could be even more effective if we did not separate the instances completely and allowed them to share some data, e.g. nodes that are common for all instances.

VI. Conclusion

In this paper we present a novel approach to detecting Skype nodes in the network. In contrast to other methods, our approach does not focus on the detection of individual Skype flows, but rather tries to reconstruct the Skype overlay network from a single seed node. The detection itself exploits the fact that if there are more Skype nodes in the network, then the sets of endpoints they communicate with will tend to overlap.

As we showed in our experiments on the University network, our detection method is very reliable and managed to detect all Skype nodes from the controlled data set. Moreover, it managed to keep a very low false positive rate, below 1%. Within the false positives, we were able to identify three main groups and for two of them we proposed methods to eliminate them and limit the false positive rate even further.

We believe this method is a good candidate for the deployment onto backbone networks because it does not inspect packets and does not calculate various statistic tests commonly used to classify Skype traffic and thus should be faster than the existing methods.

Acknowledgement

This material is based upon work supported by the ITC-A of the US Army under Contract W911NF-12-1-0028 and by ONR Global under the Department of the Navy Grant N62909-11-1-7036. Also supported by Czech Ministry of Interior grant number VG2VS/189.

References

[1] Skype Traces. http://tstat.tlc.polito.it/traces-skype.shtml. [Online; accessed 11-May-2011].

[2] D Adami, C Callegari, S Giordano, M Pagano, and T Pepe. A Real-Time Algorithm for Skype Traffic Detection and Classification. In *Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces*, pages 168–179, St. Petersburg, Russia, 2009. Springer-Verlag.

[3] S. a. Baset and H. G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–11. Ieee, 2006.

[4] Tehseen Baweja. With 663 Million Registered Users, Skype Earned $860 Million Last Year. http://techie-buzz.com/tech-news/with-663-million-registered-users-skype-earned-860-million-last-year.html, 2011. [Online; accessed 24-August-2011].

[5] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi. Detailed Analysis of Skype Traffic. *IEEE Transactions on Multimedia*, 11(1):117–127, January 2009.

[6] Dario Bonfiglio, M. Mellia, M. Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. *ACM SIGCOMM Computer Communication Review*, 37(4):37–48, 2007.

[7] Baris Coskun, Sven Dietrich, and Nasir Memon. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 131–140, New York, NY, USA, 2010. ACM.

[8] Sven Ehlert, Sandrine Petgang, and T Magedanz. Analysis and signature of Skype VoIP session traffic. In *4th IASTED International*, 2006.

[9] Saikat Guha, Neil Daswani, and Ravi Jain. An experimental study of the skype peer-to-peer voip system. In *Proceedings of IPTPS*, volume 6, pages 5–10. Citeseer, 2006.

[10] Irfan Ul Haq, Sardar Ali, Hassan Khan, and Syed Ali Khayam. What is the impact of p2p traffic on anomaly detection? In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 1–17, Berlin, Heidelberg, 2010. Springer-Verlag.

[11] Peter Parkes. 30 million people online on Skype. http://blogs.skype.com/en/2011/03/30_million_people_online.html, 2011. [Online; accessed 24-August-2011].

[12] Dario Rossi, Marco Mellia, and Michela Meo. Following skype signaling footsteps. In *2008 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks*, pages 248–253. Ieee, February 2008.

[13] P.M. Santiago del Rio, J. Ramos, J.L. Garcia-Dorado, J. Aracil, A. Cuadra-Sanchez, and M. Cutanda-Rodriguez. On the processing time for detection of Skype traffic. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1784–1788, 2011.

[14] Philipp Svoboda, E. Hyytia, F. Ricciato, M. Rupp, and M. Karner. Detection and tracking of Skype by exploiting cross layer information in a live 3G network. *Traffic Monitoring and Analysis*, pages 93–100, 2009.

[15] B. Trammell and D. Schatzmann. A tale of two outages: A study of the skype network in distress. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1282 –1286, july 2011.

[16] Brian Trammell, Elisa Boschi, Gregorio Procissi, Christian Callegari, Peter Dorfinger, and Dominik Schatzmann. Identifying Skype Traffic in a Large-Scale Flow Data Repository. *Traffic Monitoring and Analysis*, pages 72–85, 2011.

[17] Dongyan Zhang, Chao Zheng, Hongli Zhang, and Hongliang Yu. Identification and analysis of skype peer-to-peer traffic. In *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*, pages 200 –206, may 2010.