Crosstalk: A Scalable Cross-Protocol Monitoring System for Anomaly Detection

Andrea di Pietro[§] Felipe Huici[§] Diego Costantini[§] Takahide Sugita[‡] Saverio Niccolini[§]
[§]NEC Europe, Heidelberg, Germany [‡]NEC Corporation

***Abstract*—Monitoring is crucial both to the correct operation of a network and to the services that run on it. Operators perform monitoring for various purposes, including traffic engineering, quality of service, security and detection of faults and mis-configurations. However, the relentless growth of IP traffic volume renders real-time monitoring and analysis of data a very challenging problem.**

In this paper we introduce Crosstalk, a scalable and efficient distributed monitoring architecture that uses cross-protocol correlation to detect network anomalies. While applicable to a wide range of applications such as botnet detection, spam mitigation and mis-configurations, we pick a point in this application space, concentrating on VoIP attacks. We present extensive simulation results based both on generated calls and on millions of Call Data Records (CDRs) from a large VoIP operator to show our approach's performance and effectiveness.

I. Introduction

Monitoring is crucial both to the correct operation of a network and to the services that run on it. While operators perform monitoring for various purposes, one of the more important ones is detecting anomalies in the network, both in terms of attacks and mis-configurations. Indeed, reports on malicious activity such as botnets, spamming and Denial-of-Service attacks [1][2], as well as problems arising from mis-configuration of hardware and firmware [3][4] are commonplace.

Monitoring large networks in order to detect such anomalies is inherently difficult for several reasons. First, many of these anomalies require cross-protocol correlation in order to be detected. Botnets, for example, often use several protocols to coordinate activities and to carry out attacks (e.g., IRC for control and SMTP to send out spam) [5][6]. Another example where cross-protocol detection is needed is VoIP, since calls tend to be split into signaling and media traffic, as is the case with SIP and RTP.

In addition to cross-protocol correlation, monitoring needs to be done in a distributed fashion, since traffic from a particular attack or a mis-configuration may cross different monitoring points in the network. Making matters more difficult is the relentless growth of IP traffic volume, nearly doubling every two years [7]; this growth raises serious scalability issues when designing a system that not only needs to monitor large quantities of traffic in real-time, but also to aggregate results in order to provide network-wide anomaly detection.

In this paper we introduce Crosstalk, a scalable architecture that gathers data from a potentially large set of distributed monitoring probes, and performs cross-protocol correlation to detect network anomalies. While previous work has looked into the area of cross-protocol detection [8][9], it has focused on single-point solutions, and so did not scale nor could it correlate attack traffic traversing more than one monitoring point. In [10] the authors implement a distributed system, but its evaluation does not show how it would scale under heavy load and a large number of monitoring probes. SDIMS [11] presents a scalable infrastructure leveraging Distributed Aggregation Trees (DATs), but again does not evaluate its performance under heavy load nor does it look at ways of reducing messaging and data transmission overheads.

As mentioned, several anomalies can be detected using cross-protocol correlation. For the purposes of evaluating Crosstalk's scalability and performance, we pick one point in this application space and focus on SIP-based VoIP attacks. In section II we describe the general Crosstalk architecture, including its use of DATs and probabilistic data structures to achieve its performance goals; section III then describes our implementation of a VoIP attack detection application using Crosstalk's mechanisms; section IV evaluates the performance of such a system and finally section V provides conclusions and a brief description of future work.

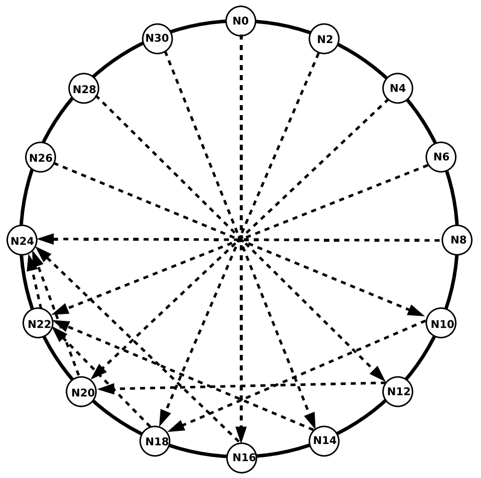
II. Crosstalk's Architecture

Crosstalk's architecture consists of three main features that allow it to perform distributed detection in a scalable way: leveraging Distributed Aggregation Trees (DATs), taking advantage of probabilistic data structures (e.g., Bloom filters), and using a novel mechanism called *backtracking* (BT).

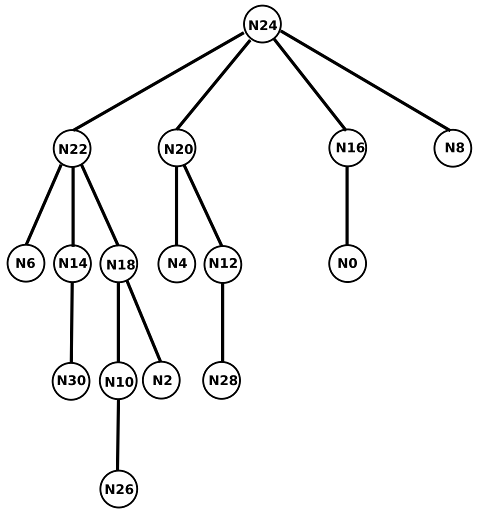
A. Distributed Aggregation Trees

The simple approach of exporting data from several monitoring probes to a centralized location clearly does not scale. In order to cope with this scalability issue, efforts both in the research and standardization communities have focused on creating tree-based hierarchies, whereby monitoring *probes* export measurements to intermediate nodes called *mediators*. These in turn perform some sort of data reduction operation (e.g., aggregating packet counts) and export the results up the tree hierarchy. In the final step the root, which is a special mediator called a *collector*, stores the aggregated results.

Ideally we would like to have a way of deriving such a tree-based topology dynamically in order to adapt to traffic conditions. This is precisely the goal of DATs, which create this structure on top of a peer-to-peer network such as Chord [12], thus providing the best from both worlds: the scalability (and resilience) of p2p networks with that provided by the aggregation mechanism of a tree structure.



(a) Chord fingers for all nodes to node N24.



(b) Same Chord fingers, this time shown as a tree.

Fig. 1. Example of a Distributed Aggregation Tree built on top of Chord with node N24 as the root.

The basic insight behind a DAT is that Chord's fingers already provide a tree structure. In order to illustrate this, figure 1(a) shows a regular Chord network with dotted lines representing the path from each node to node N24 (perhaps the responsible node for a particular key); figure 1(b) then shows these same connections but this time drawn as a tree. As can be seen, for any given key, Chord naturally builds a tree rooted at the node responsible for that key. In this way, each key has its own DAT, with all the DATs sharing the same peer-to-peer infrastructure. Within each DAT, intermediate nodes (i.e., all nodes except the leaves of the tree) can aggregate data as it travels towards the root, thus providing scalability.

B. Probabilistic Data Structures

Clearly nodes in the DAT will need to export information about the monitored data, and this will consume bandwidth. Another consideration for real-time monitoring and detection is being able to perform the cross-protocol correlation quickly. To achieve both of these goals we rely on probabilistic data structures, and more specifically Bloom filters (BFs)[1].

The details of what a Bloom filter represents are application-specific, but generally we use one filter per protocol. For instance, in the case of VoIP attacks presented later in the paper, we use one filter for SIP traffic and another one for RTP, with an entry in a filter denoting a SIP identifier such as the SIP dialog id.

This compressed representation of the data consumes little bandwidth when transmitted, and it allows us to perform aggregation and correlation (since they are based on fast bitwise operations) very quickly. Probabilistic data structures do carry a cost in the form of false negatives/positives; next we introduce a mechanism to deal with this and in section IV we evaluate their impact.

[1]While Crosstalk can function with more advanced probabilistic data structures such as sketches, Bloom filters are simpler and provide all the necessary mechanisms.

C. Backtracking

While some applications might be content to only receive the summarized data from a DAT's collector, others will use such summarized data as a trigger for retrieving more detailed information (e.g., packet headers) at the monitoring probes, perhaps to determine the cause of the trigger. In addition, Bloom filters carry a low but non-negligible probability of false positives, and so we need a way to verify whether a result is valid or just a false positive.

In order to accomplish these goals we introduce a mechanism called *backtracking*. The idea behind it is simple: when exporting Bloom filters to nodes in the DAT, keep a copy of them locally so that the system can track back to the original probes that monitored the traffic.

Figure 2 shows the process in greater detail. Probes P0 and P3 monitor traffic and export data about it in the form of Bloom filters, depicted as a set of squares with each square representing a bit in the filter (note that the figure is simplified for explanatory purposes: normally an entry in the Bloom filter would use up several bits, and more than one Bloom filter would be used to represent the protocols to be correlated). In addition, probes, as well as mediators, keep a local copy of exported Bloom filters, shown in the figure in grey. As the exported filters travel up the tree, mediators perform a bitwise operation to combine the filters, which eventually reach the collector C.

Upon receiving all the combined data, C correlates Bloom filters from different protocols and, depending on the application, triggers a backtracking request to all its immediate mediators, in this case M4 and M5. The request includes the collector's Bloom filter (shown in white), which the mediators use to compare it with their locally stored state by performing a bitwise operation: if the number of set bits in the resulting filter is higher than a user-defined threshold, the backtracking request is propagated to all of the mediator's children; otherwise, no relevant probes exist in this area of the DAT and the backtracking process finishes. Eventually the

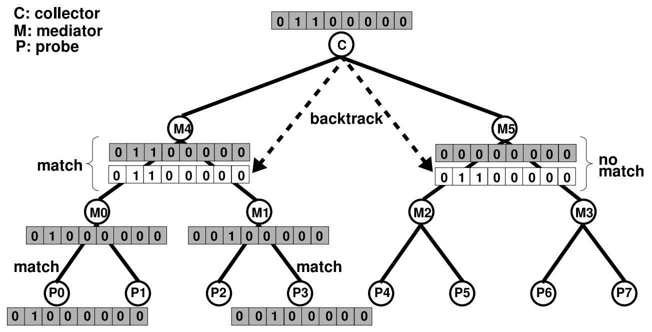


Fig. 2. Example of backtracking. Each set of squares represents a Bloom filter. A dark Bloom filter represents stored state, while a white one information sent with the backtracking request. Here the matching is done using a bitwise “AND”.

backtracking message arrives at the probes, in this case P0 and P3. In section IV we provide an evaluation of the costs associated with this mechanism and show its applicability even in large networks.

III. Application: VoIP Attack Detection

In order to evaluate the performance of Crosstalk, we implemented an application over it aimed at detecting SIP-based VoIP attacks. Several types of attacks on SIP and its related media protocol, RTP, exist. In [13], for example, the authors describe billing frauds whereby a legitimate host is sent a fake SIP BYE message causing its call to be hijacked so that the attackers are using the operator’s resources while the victim is being billed for the communications. The work in [8] describes an attack targeted at fooling the billing system into thinking a call is over by prematurely sending a SIP BYE message while keeping the corresponding RTP media traffic going. Further, SIP communications are also vulnerable to Denial-of-Service attacks such as BYE and CANCEL attacks, as well as call hijacking based on fake REINVITE messages [10].

The common thread among all of these attacks is that the RTP session keeps flowing (at least in one direction) even though the SIP control session has been torn down or redirected. As a result, Crosstalk can be used to detect such attacks by detecting a live RTP flow corresponding to a recently terminated (or redirected) SIP session. After this detection, backtracking can be used to reveal the actual nature of the anomaly, to pinpoint the malicious users, and to discard false positives. We use the remainder of the section to describe the application’s implementation details, and evaluate its performance in the next section.

Our attack detection application works as follows: each of the probes monitors all the ongoing SIP and RTP traffic. The SIP messages are parsed and, for each call, the two end-points of the media traffic are located by examining the SDP data; as for the RTP traffic, the two end-points simply correspond to the source and destination addresses of the messages.

Our method assumes the probes to synchronously and periodically export their probabilistic summaries of the monitored traffic. Thanks to the large time granularity involved in voice traffic (seconds), an offset of tens of milliseconds among the probes’ clocks is certainly acceptable; consequently, the probe synchronization requirement can be simply met by using protocols like NTP, with no need for special-purpose hardware.

The monitored data is used to fill two Bloom filters:

- A Bloom filter for keeping track of the end-points of the media traffic corresponding to SIP calls that have been terminated (or redirected) within the last measurement period.
- A Bloom filter for keeping track of the end-points of the RTP sessions that have been terminated (or redirected) within the last measurement period.

Clearly, the hash functions associated with these BFs must be the same so that the RTP and SIP endpoints associated with the same call are hashed into the same bit positions. Once these BFs are created, they are exported to the nearest mediator (i.e., the parent of the probe in the DAT). The mediator then joins all of the SIP BFs and all the RTP BFs received from its sons by performing a bitwise “OR”, thus obtaining two summarized BFs that it forwards to its own mediator. Each mediator along the way caches a copy of the last BFs it has sent up the DAT in order to support possible backtracking requests.

The detection of the anomalous behavior is achieved by a node in the DAT performing a bit-wise “XOR” of the RTP and SIP BFs: if two bits in the same position are different, that means that either the data stream or the control stream have not been terminated. In that case, all of the node’s children receive a backtracking request which includes the indices of the unmatched bits (i.e., the set bits that appeared in one BF but not the other). Each intermediate node then checks such bits against its cached aggregated BFs, and, if at least one among those is set, it propagates the BT request to its children. Such a procedure is repeated recursively until all the probes which have logged relevant information are reached.

Of course, collisions with other calls in the BFs may prevent the detection of such an event (a false negative); however, as we will show in the following sections, the system can be dimensioned in order to keep this probability arbitrarily low. On the other hand, the false positive rate is almost negligible for this system: as collisions on the BFs cannot generate false alarms, these events can happen only in very rare cases:

- When, upon detection of an actual anomalous event, the backtracking requests reach some probes which did not log relevant information
- When the terminations of the media and control flows happen so close to the boundary between two measurement intervals that the two events are recorded in different time windows.

In both cases, false alarms are easily spotted: in the former, the post-event analysis (perhaps looking at logs) allows to discard pointless requests, while in the latter it is enough to match two adjacent time windows. We point out that, in case a system cannot tolerate any missed detections, a minor modification to our system allows us to fulfill this requirement: use the RTP BF to record the end-points of the ongoing (instead of those of the terminated) media flows, and detect anomalies by looking for matching bits between the two aggregated BFs by performing a bitwise “AND”; this change comes at the the cost of increased utilization of network resources (larger BFs are in general needed).

IV. Evaluation

In this section we provide extensive simulation results to show the performance of Crosstalk, and in particular that of the VoIP attack detection application. Please note that throughout this section we use the term report to mean the Bloom filters exported between probes and mediators as a result of the monitoring and aggregation process.

A. Setup

In order to assess the performance of our solution, we evaluated several performance parameters through extensive simulations. In greater detail, we extended the Oversim overlay network simulator [14] by implementing a new application module with Crosstalk’s basic functionality and which runs on top of the Chord.

The input to the simulation consists of Call Data Records (CDRs), in order to match the format used by our VoIP data set (a CDR is a short record of a VoIP communication, including fields like caller, callee, and call duration). To have control over their distribution, CDRs are fed to the simulated monitoring probes by a centralized CDR dispatcher module: each node is assigned a given range of the overall hash ID space and each CDR is handed over to the responsible node (based on its source address).

In order to simulate the fact that RTP and SIP traffic for the same call may traverse different paths, two separate copies of the same CDR, representing in turn the RTP and SIP traffic associated with a given call, are assigned to two distinct probes by using two independent hash functions. Such a choice is rather conservative, since in a significant fraction of the real cases, the two traffic streams are likely to follow the same path, but this approach is still useful to show that our system can cope with even this extreme case. It is worth noting that the simulated probes are actually nodes on the DATs, meaning that they can act as probes for one key but as mediators (and even collector) for others simultaneously.

Regarding CDR generation, we took two approaches. First, we generated CDRs randomly by setting the timestamps and the call durations according to a Poisson process (such a simple model has been extensively used in the field of telephone traffic measurement). The purpose here was to be able to effectively tune and change the simulation parameters to show the performance of the system. In the second approach we relied on an extensive data set gathered from a large VoIP operator in order to demonstrate Crosstalk’s applicability to a real world scenario. In both cases we modified the CDRs at a certain rate (set as a percentage of the total CDRs) in order to simulate malicious calls.

B. Performance Analysis

In this section we present simulation results based on generated CDRs in order to assess the system’s performance and scalability. Crosstalk’s VoIP application depends on a number of different parameters:

- **Bloom filter size**, which affects several factors such as the missed detection rate, the bandwidth consumed and how much state nodes in the DAT keep.
- **The call rate**, in other words, how much traffic the system needs to monitor, export, and correlate.
- **The measurement interval**, which determines how long the probes keep data locally before exporting (longer intervals result in lower overheads but increase the detection delay).
- **The anomaly rate**, or percentage of malicious calls, which increases the costs associated with backtracking requests.
- **The number of probes**, equal in our case to the number of nodes in the p2p system, affecting the DAT’s topology and therefore the messaging overhead, the amount of aggregation, and the detection delay.

Bloom filter size and call rate: For the first experiment we took a look at the first two parameters and their relationship to false negatives and positives. In other words, given a certain call rate, how would an operator deploying Crosstalk dimension the Bloom filter size (which affects things like bandwidth consumption) so that the false negative and positive rates are relatively low? To this end, consider that missed detections (i.e., false negatives) happen when a collision in one BF causes a match with a “true” set bit in the other BF, resulting in the “XOR” matching operation to return 0 (the misdetection). Because the cause is the collision within a BF, all the well-known results about BF performance evaluation and dimensioning apply to our system. In particular, as the number of keys in the BFs equals the call rate λ times the measurement period T, the missed detection (md) probability can be expressed as:

$$P(md) = (1-(1-\frac{1}{M})^{K\lambda T})^K \sim (1-e^{-\frac{K\lambda T}{M}})^K$$

where M stands for the BF size (in bits) and K for the number of hash functions (which is set to an optimal value depending on the other parameters). In order to dimension the BF size for a given missed detection probability, the following inequality can be leveraged:

$$M \geq \lambda T \log_2(e) \log_2(\frac{1}{P(md)})$$

As for the false positives, they are mainly due to backtracking (BT) messages reaching probes which did not log any relevant event. However, since BT requests are triggered only when an actual anomaly is detected, the probability of such events (an anomaly happening and a request reaching a wrong probe) is definitely low. During our simulations we never observed more than a dozen such events, even when generating hundreds of thousands of calls.

In order to verify this model’s accuracy, we ran simulations to evaluate the missed detection probability for two different BF sizes and call rates ranging from 10 to 1,000 calls per second, plotting the measured results against the model's expected values (see figure 3). As shown, the model can, to a fairly high degree, predict the actual system behavior.

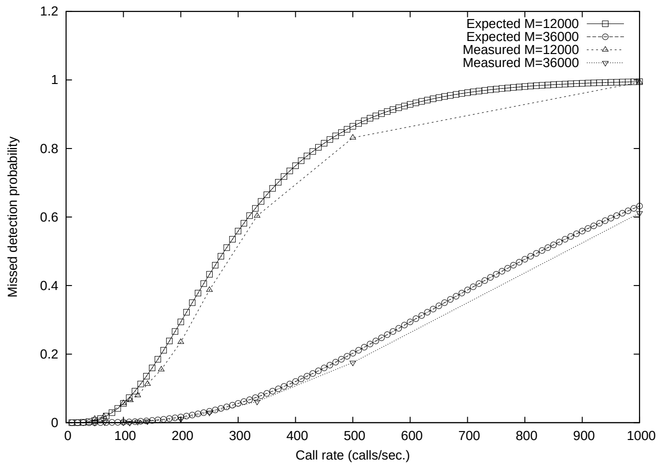


Fig. 3. Measured and expected missed detection probabilities for different call rates and BF with sizes of 12,000 and 36,000 bits.

To get a feel for the system's performance we rely on this model and on our CDR database, which shows a peak call rate well below 100 calls per second. Even if we assume that since more and more users are migrating from PSTN to VoIP such a figure will increase in the future by an order of magnitude, our system can handle the resulting traffic volume (1,000 calls/sec.): exporting data every 10 seconds and using 17KB-wide BFs yields a target missed detection probability of 10^{-3}, while using 34KB-wide BFs yields a target missed detection probability of 10^{-6}. For a measuring infrastructure made up of 1,000 probes the *total* reporting traffic adds up to only few MB/sec.

Measurement interval: If the BF size does not vary, a longer measurement period implies a larger number of keys in the BF, and, in turn, an increased missed detection probability. On the other hand, of course, this involves a lower bandwidth consumption, as data summaries are exported less frequently. Depending on the operational constraints, the previously presented mathematical model allows to find out a good trade-off; we do not present more extensive results here due to space constraints.

Anomaly rate: The next parameter we looked at was the anomaly rate, and in particular how it affects the costs related to backtracking (BT). Backtracking is triggered by either a detected anomaly or a false positive. Assuming a well-dimensioned system with a low false positive rate (e.g., less than 1%) and no anomalies, simulation results show about an order of magnitude difference between export messages and BT messages.

Arriving at more precise figures is difficult since the actual number of backtracking messages generated depends on the number of probes which have to be reached by a BT request and on the topology of the tree. Having said that, we ran a simulation to get a feel for the effects of the anomaly rate on the system, and in particular the cost of backtracking (see figure 4).

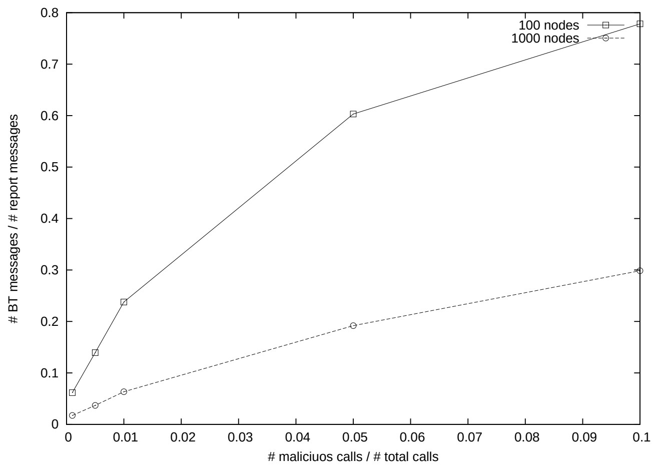


Fig. 4. Rate between backtracking and report messages in DATs made up of 100 and 1,000 probes with a varying anomaly rate.

The figure shows that, unless a very unrealistic scenario is assumed (a network where one in ten calls is malicious), the fraction of BT messages is small (usually an order of magnitude smaller) with respect to the number of reports, which proves that the BT mechanism can locate the relevant probes without flooding the DAT with messages. Further, the behavior of the system improves as the number of nodes increases.

These figures can be even further improved by reducing the size of each BT message. Observe that the Bloom filter obtained through a bit-wise "XOR" of the aggregated SIP and RTP reports must have a very limited number of set bits (in fact, the number of such bits should be lower than the number of malicious calls times the number of hash functions), which lends itself to compression. In order to effectively compress such a "sparse" bitmap, it is sufficient to include the indices of the set bits within the BT message: the resulting message size would be roughly some dozens of bytes, which is negligible when compared to the bandwidth consumed by the reporting messages. Even more efficient compression schemes for sparse bitmaps can be adopted: Fastbit [15] is just an example of a technique achieving good compression while still allowing bitwise operations to be performed over the codified data.

Number of nodes: The number of probes does not affect the accuracy of our system (that, in fact, depends on the overall number of monitored calls) but rather the aggregation and backtracking delay and the overall bandwidth consumption. The former depends on the depth of the tree, which, in turn, grows logarithmically with the number of probes. On the other hand, the overall bandwidth consumption due to the report messages grows linearly with the number of probes (each additional probe corresponds to an additional edge on the tree, which, in turn, corresponds to an additional report being transmitted). As for the BT requests, their amount depends on several variables, but we already showed their bandwidth consumption to be negligible with respect to that of the report messages.

calls/sec.	P(md)	P(fp)	no. BT/ no. reports
30	0.0014782	0.0000316	0.2585591
34	0.0014215	0.0000632	0.258106
42	0.0033482	0.0000527	0.2986333
41	0.0016393	0.0000632	0.2970421
33	0.0021536	0.0000738	0.2561671
8	0.0000000	0.0000000	0.1249855
1.3	0.0000000	0.0000000	0.024015
0.5	0.0000000	0.0000000	0.0125396

Fig. 5. Crosstalk's performance when using real-world data from a large VoIP provider. Each row represents a different 30-minute time sample in our data set.

C. Real-World Performance

In the previous section we looked at how Crosstalk behaves when varying a number of different parameters in simulation. To get a sense of how it would perform in a realistic scenario we replaced the generated CDRs with those of an extensive data set consisting of more than 100 million CDRs from more than 15 million users collected over a period of more than 4 weeks. In order to test our system in different traffic scenarios without having to face prohibitive simulation times, we ran our experiments by using 30 minutes time slots that we sampled out of our complete database. We assumed a system with 1,000 probes, 4KB BFs, a measurement interval of 20 seconds, and a (quite conservative) anomaly rate of 3% (see figure 5). The results clearly show that Crosstalk is more than able to cope with this traffic, yielding very low false negative (md) and false positive rates as well as a negligible number of backtracking messages with respect to the number of reports. It is worth noting that while the table lists figures from a few 30-minute time samples in our data set (one per row), we ran simulations for others and obtained similar results.

V. Conclusions and Future Work

We have presented Crosstalk, a scalable and distributed monitoring system for detecting cross-protocol anomalies. We have implemented a VoIP attack detection application over it and presented extensive simulation results on a large VoIP data set. In addition, we used a mathematical model to show that Crosstalk performs well even when presented with much higher loads than those conveyed by current VoIP infrastructures.

The results confirm that Crosstalk can scale to a very large number of monitoring probes, deal with a large call rate of 1,000 calls/sec and a high percentage of anomalous calls, all while using small Bloom filter sizes of only dozens of KB. The system can be easily tuned to achieve arbitrarily small missed detection rates with a limited increase in terms of overhead. Moreover, in case missing an anomaly is not acceptable, a slight change in the system layout allows Crosstalk to fulfill such a requirement.

One of the topics we did not discuss due to space constraints is tree topologies. The DATs we used relied on Chord's normal routing algorithm, which can result in unbalanced trees, specially when the DAT contains a large number of nodes. Towards a solution, previous work [16] modified Chord to provide (almost) balanced binary trees. While certainly an improvement, what we would like is not only to have a mostly balanced tree, but also the ability to control its depth; in other words, controlling the trade-off between scalability through aggregation (achieved with deeper trees) and aggregation delay and messaging overheads (reduced by using shallower trees). We are currently working on an algorithm to achieve this.

VI. Acknowledgements

This work was partly supported by the Japanese Ministry of Internal Affairs and Communications (MIC).

References

[1] The Economist, "A walk on the dark side," http://www.economist.com/displayStory.cfm?story_id=9723768, August 2007.

[2] Symantec Corporation, "Internet Security Threat Report Volume XI," http://www.symantec.com/enterprise/threatreport/index.jsp, March 2007.

[3] McPherson, D., "Internet Routing Insecurity: Pakistan Nukes YouTube?" http://asert.arbornetworks.com/2008/02/internet-routing-insecuritypakistan-nukes-youtube/, February 2008.

[4] Plonka, D., "Flawed routers flood university of wisconsin internet time server," http://pages.cs.wisc.edu/ plonka/netgear-sntp/, August 2003.

[5] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *SS'08: Proceedings of the 17th conference on Security symposium*. Berkeley, CA, USA: USENIX Association, 2008, pp. 139–154.

[6] W. T. Strayer, D. Lapsley, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet Detection: Countering the Largest Security Threat*, W. Lee, C. Wang, and D. Dagon, Eds. Springer-Verlag, 2007.

[7] Cisco Systems, "Approaching the zettabyte era," "http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481374.pdf", June 2008.

[8] Y.-S. Wu, S. Bagchi, S. Garg, N. Singh, and T. Tsai, "Scidive: A stateful and cross protocol intrusion detection architecture for voice-over-ip environments," in *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2004, p. 433.

[9] B. Barry and A. Chan, "Towards intelligent cross protocol intrusion detection in the next generation networks based on protocol anomaly detection," in *The 9th International Conference on Advanced Communication Technology*, 2007, pp. 1505–1510.

[10] Y.-S. Wu, V. Apte, S. Bagchi, S. Garg, and N. Singh, "Intrusion detection in voice over ip environments," *International Journal of Information Security*. [Online]. Available: http://dx.doi.org/10.1007/s10207-008-0071-0

[11] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," in *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2004, pp. 379–390.

[12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2001, pp. 149–160.

[13] R. Zhang, X. Wang, X. Yang, and X. Jiang, "Billing attacks on sip-based voip systems," in *WOOT '07: Proceedings of the first USENIX workshop on Offensive Technologies*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–8.

[14] I. Baumgart, B. Heep, and S. Krause, "OverSim: A Flexible Overlay Network Simulation Framework," in *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007, pp. 79–84.

[15] "Fastbit: An efficient compressed bitmap index technology." [Online]. Available: http://sdm.lbl.gov/fastbit/

[16] M. Cai and K. Hwang, "Distributed Aggregation Algorithms with Load-Balancing for Scalable Grid Resource Monitoring," *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 123, 2007.