Massive Scale Cyber Traffic Analysis: A Driver for Graph Database Research

Cliff Joslyn[*]
National Security
Pacific Northwest National Laboratory (PNNL)
cliff.joslyn@pnnl.gov

Sutanay Choudhury
Fundamental Sciences
PNNL
sutanay.choudhury@pnnl.org

David Haglin
Fundamental Sciences
PNNL
david.haglin@pnnl.gov

Bill Howe
Computer Science & Engineering
University of Washington
billhowe@cs.washington.edu

Bill Nickless
National Security
PNNL
Bill.Nickless@pnnl.gov

Bryan Olsen
National Security
PNNL
bryan.olsen@pnnl.gov

ABSTRACT

We consider cyber traffic analysis (TA) as a challenge problem for research in graph database systems. TA involves observing and analyzing connections between clients, servers, hosts, and actors within IP networks, over time, to detect suspicious patterns. Towards that end, NetFlow (or more generically, IPFLOW) data are available from routers and servers which summarize coherent groups of IP packets flowing through the network. The ability to cast IPFLOW data as a massive graph and query it interactively is potentially transformative for cybersecurity, but issues of scale and data complexity pose challenges for current technology. In this paper, we outline requirements and opportunities for graph-structured IPFLOW analytics based on our experience with real IPFLOW databases. We describe real use cases from the security domain, cast them as graph patterns, show how to express them in two graph-oriented query languages (SPARQL and Datalog), and use these examples to motivate a new class of "hybrid" graph-relational systems.

Keywords

Graph databases, cyber-security, traffic analysis.

1. INTRODUCTION

It is widely recognized that cyber technologies represent one of, if not the most, significant challenges to national security today. Cybersecurity analysts must increasingly manipulate massive-scale, high-resolution flows to identify, categorize, and mitigate attacks involving networks spanning institutional and national boundaries. A flow in this context is an aggregation of packet-level communication between two cyber systems over some network protocol between specific ports for a period of time. Flow data can be acquired at all levels of the Internet Protocol (IP) communications hierarchy, starting with bits and packets: Each packet passing through a router or switch is inspected for a set of attributes and determined to be unique or associated with packets already seen. All packets with the same source, destination, ports and protocol are grouped into a flow and packets and bytes are aggregated accordingly. We find that this level of aggregation at the flow level is distinctly valuable and complementary to the more detailed packet level for exposing and reasoning about the communication graph patterns with which analysts work.

Analysis of these graph-structured flows represents an excellent use case for advancing database research, for several reasons: 1) the problem is ubiquitous in enterprise and government computing allowing solutions to have enormous direct impact; 2) representative large-scale datasets suitable for reproducible research are available; 3) datasets of nearly arbitrary size are easy to collect; 4) the complexity of these datasets motivates new architectures and algorithms; 5) the analysis tasks are much more complex than the simple graph-oriented benchmarks and use cases proposed in the literature (e.g., counting triangles or computing the degree distribution); and 6) there is a rich literature to draw on from the networking and computer security community involving IPFlow data and algorithms.

While there have been a number of past attempts at IPFLOW graph analytics, including by us [10], they still remain limited in scale or complexity [3, 4, 5, 6, 8], and frequently do not consider flow direction or other attributes [2, 9] (although Tegeler et al. [12] do consider temporal information).

2. IPFLOW GRAPHS

Table 1 shows a typical data schema for IPFLOW records.[1] In addition to source and destination IP address and port, the number of packets and bytes comprising the flow, start and stop time, transport protocol (e.g. TCP vs. UDP), and other flags are included. This schema is naturally repre-

[*]Corresponding author: 1100 Dexter Ave. N., Suite 400, Seattle, WA 98109, 206-552-0351.

[1]We are limiting our work to IPv4 but consider support for IPv6 within reach of our techniques.

Proceedings of the First International Workshop on Graph Data Management Experience and Systems (GRADES 2013) June 23, 2013, New York, NY, USA
Copyright 2013 ACM 978-1-4503-2188-4 ...$15.00.

Field	Type	Description
EXADDR	BIGINT	IP address of recording device
DPKTS	BIGINT	Destination packet count
DOCTETS	BIGINT	Destination byte count
STIME	BIGINT	Flow start time
FTIME	BIGINT	Flow end time
SRCADDR	BIGINT	Source IP address
DSTADDR	BIGINT	Destination IP address
SRCPORT	INTEGER	Source port
DSTPORT	INTEGER	Destination port
PROT	INTEGER	Transport protocol
TOS	INTEGER	Type of service
TCP_Flags	INTEGER	Header flags

Table 1: Spec for input data set.

sented as a directed graph (sample edge shown in Fig. 1), but with additional complexities, including:

- A combinatorial structure on node IDs, each a vector of the four "octets" of the IP address, together with the port ID.
- # packets and # bytes as quantitative edge attributes.
- An interval attribute for start and stop times.
- A categorical attribute for transport protocol.

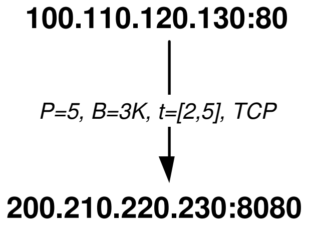


Figure 1: Link in an IPFLOW graph: nodes are IP:Port, P=# packets, B=# bytes, t=time interval. Transport protocol also shown.

Under this model, IPFLOW graphs can be considered hybrid structures involving both graph components and relational components, potentially motivating a new class of systems. Our use cases below exercise the hybrid aspects of these datasets to detect events of interest. In very large databases, the ability to handle (potentially recursive) connectivity queries combined with conventional relational data processing is seeing increasing attention, for example in the Graphcube effort [13].

3. DATA SETS

Here we review the characteristics of IPFLOW data sets modeled as graphs and their utility as a "challenge problem" for graph databases, including scale and acquisition issues. We argue that appropriate datasets of any size are readily available for research purposes either by direct acquisition or by reuse of publicly available sources. We describe some specific, prominent public datasets and characterize their graph-theoretic properties.

Organizations like PNNL collect IPFLOW data for multiple purposes and levels. IPFLOW data are readily obtained by system and network administrators, and individual users, both on individual network nodes, network appliances, at organizational boundaries, central ISP routers, and at other points in a network architecture. For example, if the "capture device" is placed at the perimeter of the network (typically a firewall), data will be collected on communications from the outside in, and from the inside out. That configuration will not provide data on communications to and from internal devices that may be valuable as well. In that case, having IPFLOW collected on internal routers and switches becomes important. These different implementations may produce vastly different amounts of data.

The size and the dynamic properties of these IPFlow graphs pose major challenges from a graph data management perspective. A typical data collection rate at one sensor can range into tens of megabits per second (see e.g. Fig. 2), implying a data volume of hundreds of gigabytes per day from one sensor. The corresponding count for number of packet-level records ranges into hundreds of millions.

Because of these factors, and also the ability to adjust both sampling rates and sampling windows, gathered data sets and corresponding IPFLOW graphs can be of highly variable sizes, with giga- and tera-scale datasets being quite common, especially considering the amount of near-continuous background communication between networked devices. As a result, an enterprise setting may produce hundreds of billions of IPFLOW records per day. Although the volume of information tends to exhibit a periodicity due to typical working hours, the continuous background communication remains even in the absence of human activity, and unusual events may occur at any time.

	Average	Stdev
Flows/day (M)	613.2	242.5
Packets/day (B)	27.6	11.9
Packets/flow	178.7	702.6
Bytes/day (T)	24.1	11.1
Bytes/flow (K)	153.1	596.4

Table 2: Flow collection statistics for a typical PNNL network monitor.

Table 2 shows collection statistics for a typical network monitor at PNNL aggregated daily. The moderately high standard deviation around the 613.2M collected flows/day reflects the weekday/weekend work schedule. It is easy to see that IPFLOW graphs of most any desirable size from the mega- to the tera-scale can be generated with ease.

IPFLOW data sets are typically sensitive and held closely by particular organizations and enterprises; they are difficult to obtain for research purposes. Some research organizations have industry or government partners who are willing to share network data for the purposes of research, but that data is unable to be shared outside of the project context and therefore hinders scientific reproducibility. Realizing this challenge, there have been several efforts to provide relevant network communication and cyber attack data for the purposes of research, for example the Cooperative Association for Internet Data Analysis (CAIDA)[2] and the Protected Repository for the Defense of Infrastructure Against Cyber

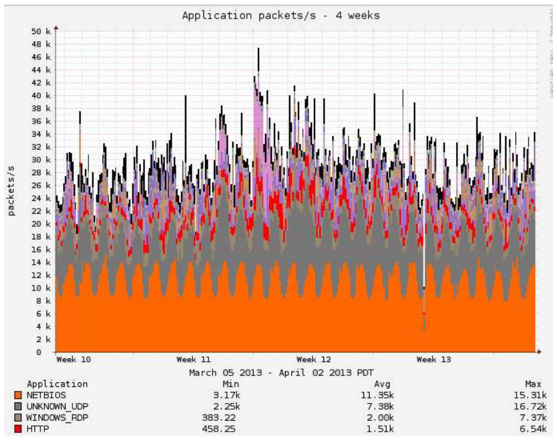
[2]http://www.caida.org

Threats (PREDICT)[3] effort by the Department of Homeland Securities' Science and Technology Directorate.

The datasets from CAIDA can be put into two broad categories, 1) active measurement of macroscopic internet topology and 2) passive measurements of internet traffic data. The former involves actively probing IP-level paths to observed IP addresses, performing alias resolution to resolve which IP addresses are associated with the same physical host, and finally yield router level internet graphs. The latter involves sampling the traffic flowing through internet service provider (ISP) routers and observing the traffic destined to specific internet address spaces.

These data are especially appealing from a graph database perspective. As various events in the real world are reflected in the network traffic activity, interactive, scalable, ad hoc query is a necessity to detect a change in the properties of a vertex or emergence of a pattern as the situation unfolds.

As an example, Fig. 3 shows the study by King and Dainotti [7] where they discovered the changes to the internet infrastructure in Egypt as the political events unfolded. Modeling the IPFlow data as a graph and querying for changes in the macroscopic properties such as degree distribution, or the in or out degree of important vertices in the graph, can be the first step in performing such analysis.



Application	Min	Avg	Max
🟧 NETBIOS	3.17k	11.35k	15.31k
⬛ UNKNOWN_UDP	2.25k	7.38k	16.72k
▨ WINDOWS_RDP	383.22	2.00k	7.37k
🟥 HTTP	458.25	1.51k	6.54k

Figure 2: Data rate (bits and packets per second) observed by the UCSD Network Telescope.

Consider a dataset collected from CAIDA[4] containing 56 million packet level records collected from a bi-directional internet backbone link located at an Equinox[5] datacenter in San Jose, CA, USA. The data collection monitor dropped less than 1% of packets in a test environment. We refer the reader to the CAIDA website for exhaustive description of the data collection methodology. Fig. 4 show a rendering of a small portion of this dataset following an aggregation to netflow level and subsequent conversion to a graph format.

While background rates can be estimated, the highly varying nature of the data poses critical graph database design questions. The number of packets and flows varies sharply depending on the events taking place, for example attack events. As an example, a prominent news source such as www.cnn.com can receive an extra-ordinary amount of traffic both due to a major news release or a DoS attack [8]. The CAIDA DDoS attack 2007[6] provides approximately one hour of anonymized traffic traces from a DDoS attack, which attempts to the block access to the targeted server by overloading the server with queries. The total size of this dataset is 21 GB and Fig. 5 shows the rise in the data volume and number of edges in the graph as the attack evolves.

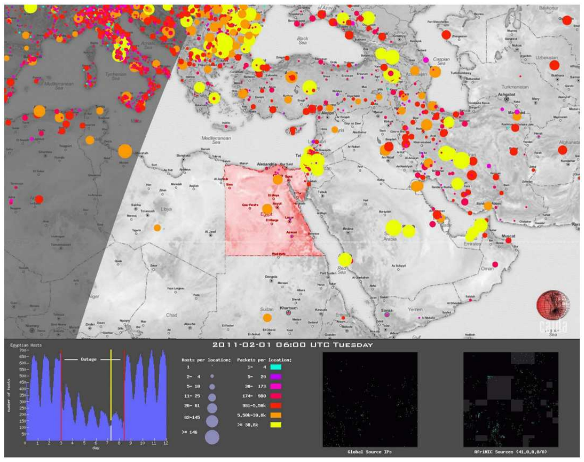


Figure 3: Visualization of the IPFlow data collected during the Egypt internet blackout [7].

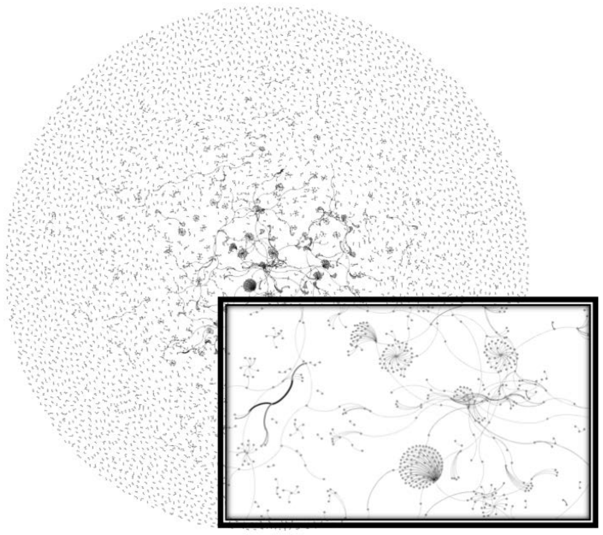


Figure 4: Internet backbone traffic graph, 10K records.

4. USE CASES

There are a number of widely known cyber attack scenarios which are ammenable to treatment in quantitatively labeled graphs like our IPFLOW graph (see also [3]). Each scenario corresponds to one or more queries appropriate for a database system. We describe representative scenarios, then dwelling in more depth on two in particular: Exfiltration and Hierarchical Botnet.

[3]predict.org

[4]www.caida.org/data/passive/passive-2012-dataset.xml

[5]www.equinox.com

[6]www.caida.org/data/passive/ddos-20070804-dataset.xml

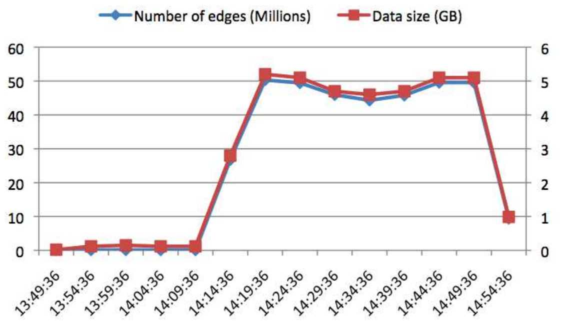


Figure 5: Statistics on graph data volume as collected from a DDoS attack dataset.

In all cases we highlight the hybrid graph queries involved, combining graph connectivity information with quantitative attribute aggregation. Below for convenience we combine # packets and # bytes in a flow to a single size attribute S.

- Watering Hole: In this scenario, a cyber adversary successfully compromises an intermediate target (e.g. the web site of a local newspaper) that is routinely accessed by computers associated with the adversary's primary target. The adversary puts malicious software on the intermediate target which, when accessed by targeted computers, causes them to directly communicate with a third machine completely controlled by the adversary. Given flow records at the perimeter of the primary target, defenders can look for a graph pattern of activity where individual accesses to a popular web site are suddenly followed by accesses to an external service—particularly an external service that has been rarely or never accessed before. Fig. 6 shows this attack as a hybrid graph pattern. Over the time interval $[1, 100]$ baseline bidirectional flow traffic is established between a bait machine and multiple targets. Then at a subsequent time, and much narrower window $[110, 120]$, all the target machines initiate a flow to a common controller.

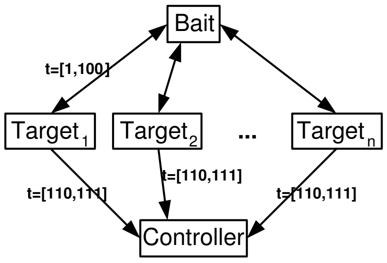


Figure 6: A watering hole attack graph pattern.

- Botnets For DDOS: Here on the order of a hundred thousand robot machines distributed widely across the Internet suddenly start sending large volumes of network traffic (either in terms of flow size or number of flows) to a relatively small number of targets, while communicating with a small number of controller hosts. Fig. 7 shows this attack as a hybrid graph pattern. A relatively small number k of controllers have relatively small bidirectional flow patterns with a large number n of bots, which in turn have very large one-directional flows with a relatively small number m of targets.

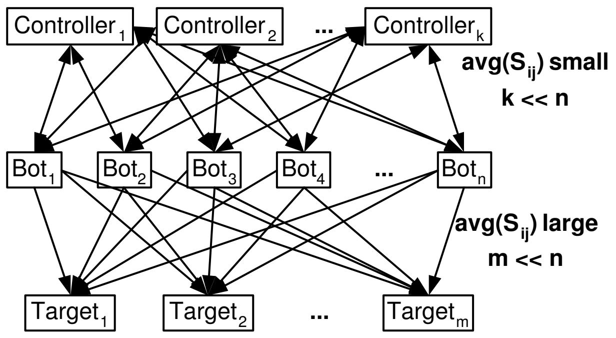


Figure 7: A botnet for DDOS graph pattern.

- Exfiltration (EXFIL): In this attack pattern, an attacker wishes to transmit a large volume of data from a target to an Internet host (drop box) under the adversary's control. The adversary uses a relatively low-bandwidth control channel to issue commands prior to a large data flow from a target host to the drop box. Given flow records at the perimeter of the primary target, defenders look for a pattern of activity where an unusually large amount of data is transmitted by a target host that does not generally transmit large amounts of data to external hosts. Additionally, defenders want to find the low-bandwidth control channel used by the adversary to initiate the outbound large data transfer.

  Fig. 8 shows this attack as a hybrid graph pattern. A relatively small number flow between attacker and target precedes (with time interval $t = [-10, 0]$) a subsequent (with time interval $[1, 3600]$) large flow (or aggregate flows) from target to dropbox. This case will be elaborated below.

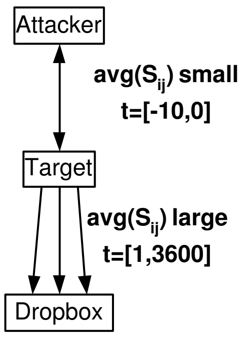


Figure 8: An exfiltration graph pattern.

- Hierarchical Botnets (HIERBOT): To hide their identity and to support larger botnets, the botnet creators and operators may establish a hierarchy of controllers. Bots talk to first level controllers, which in turn talk to second level controllers, and so on. Internet Service Providers must identify all the controllers (not just the first level in the hierarchy) in order to disrupt the botnet control structure. The most interesting controllers are the highest in the hierarchy, as they are more likely to be associated with the people that create and operate the botnet. Fig. 9 shows this attack as a hybrid graph pattern, showing a tree pattern of controllers arranged in p levels each with k_p sub-controllers per level. It is basically an elaboration on Botnet for DDOS, with the crucial property that paths can be of indefinite length, requiring recursive graph pattern finding. This case will be elaborated below.

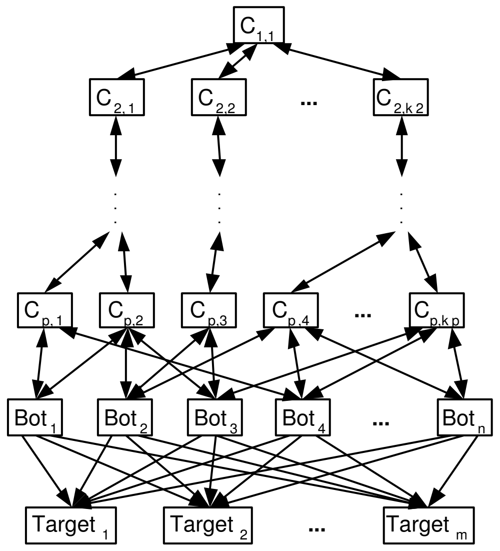


Figure 9: A hierarchical botnet graph pattern.

5. USE CASES CAST AS GRAPH QUERIES

Having framed use case scenarios as patterns in quantified IPFLOW graphs, we now elaborate on the two cases of EXFIL and HIERBOT by showing translation into two graph query languages: SPARQL and Datalog. These languages are shown as exemplars of graph query languages. By elaborating on these two example use cases and languages, we demonstrate the power of the graph database ad hoc query mechanism as tool in support of these cyber use cases.

5.1 SPARQL

The SPARQL suite of standards[7] consists of languages and protocols to query RDF graphs made up of triples (subject, predicate, object). Triples represent edges in a directed graph, where the names of the nodes in the graph are the strings appearing as either subject or object in at least one of the triples, and the edges are typed by the predicates. The SPARQL query language is modeled after SQL, with the SELECT clause requesting graph patterns that might appear in the larger data graph (perhaps in multiple places of the graph). There are several open source and commercial products that operate in the RDF/SPARQL space, including Virtuoso, AllegroGraph, uRiKA, Jena/ARQ. We now consider the use of SPARQL for the Exfiltration use case and Datalog for the Hierarchical Botnets use case.

The SPARQL query in Figure 10 consists of three components: the first for finding the control message, the second for pulling out the large amount of data from victim to dropbox within an hour of receiving the control message, and the third for considering the highest outflow of data from the victim in the previous hour and ensuring that is not very much. While the check for the previous hour having very little outflow of data may not be sufficient for a true cyber query, this example shows a way to formulate a reasonably complex query in a short and expressive way.

[7]http://www.w3.org/TR/sparql11-overview

```
SELECT ?control ?target ?dropbox ?xfil WHERE {
 # Control Message from C2 to target
  ?control ?ctrlmsg ?target .
  ?ctrlmsg :FTIME ?ftime1 .
  ?ctrlmsg :STIME ?stime1 .
  ?ctrlmsg :DPKTS ?pkts1 .
  ?ctrlmsg :DOCTETS ?octets1 .
  FILTER (?pkts1 < 3 && ?octets1 < 300)

  # xFil occurs within the next hour to ?dropbox
  { SELECT ?target ?dropbox (SUM(?octets) AS ?xfil)
    WHERE {
      ?target ?flow ?dropbox .
      ?flow :DOCTETS ?octets .
      ?flow :STIME ?stime .
      FILTER (?stime > ?ftime1
              && ?stime - ?ftime1 < 3600)
    } GROUP BY ?target ?dropbox
      HAVING (SUM(?octets) > 200000)
  }

  # xFil did NOT happen from target in previous
  # hour (target usually does not send lots of
  # data to external hosts).
  { SELECT ?target
    { SELECT ?target (SUM(?octets) as ?outRate)
      WHERE {
        ?target ?flow ?dst .
        ?flow :DOCTETS ?octets .
        ?flow :STIME ?stime .
        FILTER (?stime < ?stime1
                && ?stime1 - ?stime < 3600)
      } GROUP BY ?target ?dst
    } GROUP BY ?target
      HAVING (MAX(?outRate) < 100000)
  }
}
```

Figure 10: SPARQL query for the EXFIL use case.

Note that there are several "threshold" constants in the query that could be tuned to a specific dataset and exfiltration scenario: the size of the control message < 300 bytes, the duration of the exfiltration message as 3600 (the number of seconds in one hour), the size of the total outflow to the dropbox of $\geq 200{,}000$ bytes, and the size of the largest outflow to another host indicating "does not generally transmit large amounts of data" of $\leq 100{,}000$ bytes.

We also note that the select variables at the outer-level select clause will show all possible dropbox hosts involved in a single exfiltration event from one victim.

5.2 Datalog

The Datalog language is a non-turing-complete subset of Prolog that extends the relational algebra with recursion, and is strictly more expressive than SPARQL[1].[8] A datalog program is a set of rules, where each rule consists of a

[8]The relationship between datalog and SPARQL 1.0, which includes recursive path expressions, is an open problem to the best of our knowledge.

```
# syntactic sugar to ignore irrelevant attributes
flow(src,dst,start,end) :-
   Ipflow(_,_,_,_,start,end,src,dst,_,_,_,_,_,_)

# count the flows to dst starting in a given window
flowcount(dst, count(src)) :- flow(src,dst,start,end),
            4:59 pm < start, start < 5:00 pm

# a DDOS attack is a large number of flows
# to one destination in a short time
DDOS(dst) :- flowcount(dst, number_of_flows),
            number_of_flows > 100k

# A bot is any IP participating in an attack
bot(src) :- DDOS(dst), flow(src, dst, start, end)

# a controller is anyone connected to a bot
contro(src) :- flow(src, dst, start, end), bot(dst)
# ...or anyone connected to another controller
contro(src) :- flow(src, dst, start, end), contro(dst)

# master controllers connect to many controllers
master(src) :- flow(src,dst,start,end),
              flowcount(dst, number_of_flows),
              contro(dst), number_of_flows > 20
```

Figure 11: Datalog code for HIERBOT.

head and a body. The body of a rule may refer to its head; in this case, the rule is recursive, although recursion can arise in other ways: if we induce a dependency graph by constructing an edge between the head of a rule and each term in its body, any cycle in this graph indicates recursion. Datalog has recently enjoyed a resurgence as a natural language for graph computations [11], thanks to its simplicity, expressiveness, formal semantics, and affordance of optimized, parallel evaluation using techniques developed for relational query processing.

Fig. 11 shows the datalog code for hierarchical botnet, highlighting the recursive nature of the path query involved. Some non-standard syntax is used for clarity. The first rule simply projects out unneeded fields. The second rule counts the number of flows in a particular time range. The third and fourth rules define a DDoS attack as any high capacity flow to a particular destination, and identifies those IPs participating in the attack, respectively. The fifth and sixth rule recursively defines a controller as any IP that is communicating with a bot, or anyone that is communicating with a controller. The final rule identifies controllers that are influencing a large number of other controllers, considering them to be potential sources of the attack.

6. ACKNOWLEDGMENTS

A part of this work was funded by the Center for Adaptive Super Computing Software (CASS) at the U.S. Department of Energy's Pacific Northwest National Laboratory. Different portions of this work were funded by the Asymmetric Resilient Cybersecurity Initiative at PNNL. PNNL is operated by Battelle Memorial Institute under Contract DE-ACO6-76RL01830.

7. REFERENCES

[1] R. Angles and C. Gutierrez. The expressive power of sparql. In ISWC, pages 114–129, 2008.

[2] B. Coskun, S. Dietrich, and N. Memon. Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts. In Proc. ACSAC 2010, pages 131–140, 2010.

[3] J. R. Goodall, W. G. Lutters, P. Rheingans, and A. Komlodi. Preserving the big picture: visual network traffic analysis with tnv. IEEE Wshop. on Visualization for Computer Security VizSEC 05, pages 47–54, 2005.

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

[5] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (tdgs). In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, IMC '07, 2007.

[6] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffic calssification in the dark. In SIGCOMM 05, 2005.

[7] A. King, A. Dainotti, B. Huffaker, and K. Claffy. A Coordinated View of the Temporal Evolution of Large-scale Internet Events. In Workshop on Internet Visualization (WIV), Nov 2012.

[8] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage. Inferring internet denial-of-service activity. ACM Trans. Comput. Syst., 24(2):115–139, May 2006.

[9] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: Finding p2p bots with structured graph analysis. In Proc. 19th USENIX Conf. on Security, 2010.

[10] J.-P. Navarro, B. Nickless, and L. Winkler. Combining cisco netflow exports with relational database technology for usage statistics, intrusion detection, and network forensics. In Proc. 14th Systems Administration Conf. (LISA 2000), pages 285–290, 2000.

[11] J. Seo, S. Guo, and M. S. Lam. Socialite: Datalog extensions for efÞcient social network analysis. In ICDE, 2013.

[12] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In Proc. Co-NEXT 12, pages 349–360, 2012.

[13] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: On warehousing and olap multidimensional networks. In SIGMOD 2011, 2011.