GULv3—A Novel Tool for Network Managers to Audit Networks

Robert F. Erbacher
Northwest Security Institute
15127 NE 24TH Street, #241
Redmond, WA 98052
Robert.Erbacher@gmail.com

Rian Shelley
Dept. of Computer Science, UMC 4205
Utah State University
Logan, UT 84322
rian.shelley@usu.edu

ABSTRACT

In any network, the network manager must know the status of the network at any given time and location. It is necessary for them to be able to track down compromised boxes and be aware of the network topology. This becomes difficult in large networks with dynamically changing connections and users, especially with extensive network information to sort through. Thus, it is challenging to locate a compromised system given nothing more than an IP address and timestamp. Our solution is a novel set of capabilities providing a single place to query the network with the dynamic content integrated. GULv3 (Grand Unified Lookup Version 3) is novel due to two reasons—the mapping technique used to map layer two and layer three devices and the GUI used to present the large amount of information. GULv3 has been tested and deployed and its performance under various constraints has been measured.

Keywords

Network Management, Security Management, User Interfaces, Attack and Compromise Identification and Tracking

1. Introduction

Network management requires a large amount of knowledge about the status of the network. This allows network managers to minimize the effect of failures or overloads, handle network reconfigurations, identify and resolve misbehaving systems, etc. This must be done in a modern networking environment in which devices, especially mobile devices, may be constantly added and removed from the network. In particular, knowledge about the network

addresses used, access time, and topology are particularly important. In any network, knowledge about topology is required in order to manage the network efficiently [3]. However, manually keeping track of the topology and changes made to it is unfeasible in any realistic network.

In a network composed of managed devices, much of the required data can be obtained from SNMP [5] queries. Additional important data can be acquired from other databases, or from the analysis of existing data. In particular, layer 2 devices can be mapped, and its topology derived, using a couple of methods, such as LLDP, MAC address tables, or spanning trees. The layer 3 topology can be mapped using trace routes, subnet locations, or route tables. The layer 2 topology can be merged with the layer 3 topology to generate a complete topology of the network. A simple interface that provides access to all of this data is what is needed. GULv3 (Grand Unified Lookup Version 3) makes use of a unique mapping mechanism to map layer two devices, which is then merged with the layer 3 mapping. In addition, GULv3 has a simple and clean GUI design that enables the tracking and displaying of the required information in an easy to use manner.

As an example of the goals of GULv3, consider the scenario of attempting to locate, isolate, and repair a compromised system. Typically, the goal is to identify what machine is compromised, who the owner of the machine is, where the system is or was connected to the network, and disallow the machine from connecting to the network until it is rebuilt; i.e., this may mean identifying the network switch port to which the system is connected and disabling the port or disabling access from the identified MAC address. Normally, this requires correlating information from multiple data sets and low-level analysis to identify where the system is connected to the network. Usually, all the network managers will have is an IP Address. The goal is to allow the IP address to be searched for within GULv3 yielding either nothing, which tells us that the IP address was spoofed during the attack, or if the IP address

is registered, it yields contact information and ARP information that tells us the MAC address being used during the time of the attack. Running a subsearch on the MAC address tells us exactly which switch port the computer was plugged into, and likely contact information.

2. Information obtained from each network layer and their mappings

GULv3 makes use of information from many different sources, namely:

- Databases
- Queries on network devices
- Previously existing information

2.1 Layer 1

For our purposes, we can obtain information about the type of link (fiber optic, Ethernet, wireless, other) from the SNMP ifType MIB (Management Information Base) [15][9][25][21][8]. We can obtain the information about switch port to wall jack mappings from a database maintained by the people who installed the equipment. While useful, this information may not be necessary and has therefore not been pursued.

2.2 Layer 2

Information about VLANs can be obtained from SNMP queries of layer two devices using the dot1qVlan subtree of the bridge extensions MIB [12]. Information about neighboring switches found by discovery protocols can be obtained via the IETF physical topology MIB [2][11]. Information about the current spanning tree topology can be obtained using the dot1dStp group of the bridge MIB. The MAC address-forwarding table can also be obtained using the dot1dTpFdb table of the bridge MIB.

2.3 Layer 3

IP routing paths, and to a lesser degree, topology, can be obtained using traceroutes [10][4]. IP addresses and subnet masks for them can be found using the ipAddressTable or the ipAddrTable of the IP MIB [23]. Discovery methods, such as discovery protocols [11] and

even internal routing protocols like OSPF [17] or ISIS [19] can tell you who they think their neighbors are. Routing tables are available via ipRouteTable in the IP MIB [23].

2.4 Other Data

Other pieces of information were useful in the development of GULv3. Registered MAC addresses and contact information could be made readily available by access to registration databases. The first three bytes of a MAC address can be used to identify the manufacturer of a network card [13][14]. This can easily be defeated by spoofing the MAC address, but our goal is the creation of a management tool and not specifically a security tool. Used bandwidth could be obtained via the ifTable in the interfaces MIB [16], or from another source that collects it such as cricket or MRTG [1][18]. Flow information can be exported from some types of routers [21], or using the FProbe software [30]. SFlow [20] packets can be exported from most managed networking devices. Most of these sources of information are useful, but are outside of the initial scope of GULv3.

2.5 Layer 2 mapping

Discovery protocols, spanning tree, and the MAC forwarding table can be used to discover the topology of a layer 2 network.

2.5.1 Discovery Protocols

Discovery protocols, such as LLDP [12] or CDP [7], are used to find nearby network devices. However, most of them are either proprietary, or relatively new. Thus while useful, they may not be used widely enough to be sole providers of the needed information.

2.5.2 Spanning Tree

The spanning tree that is designated the root field in the dot1dStp MIB [12] can be used to find the topology as well. It is as simple as creating a list of edges between the current switch and the designated root, wherever the designated root is not the current switch. This has the advantage of also displaying links that have been shut down. This will allow the extraction of the real world graph from the switches. If there is a mismanaged switch, it is still possible to

find its MAC address if it happens to be the designated bridge of a link. However, in this case, the resulting topology may be incomplete.

2.5.3 MAC Forwarding Tables

If the MAC address-forwarding table is available, a topology can be built from the bottom up using the algorithm below. We developed this algorithm so that there is a method to discover the topology when discovery protocols are not available; it is this algorithm that appears in GULv3.

Consider a switch N1 as a node on the graph T. The set of ports for the switch is p ∪ u, where p is the set of ports that are connected to other switches, and u is the set of ports connected to users, or not used at all. p ∪ u is what is obtained when the switch is queried. We can remove the set p from p ∪ u by obtaining the MAC addresses used by each switch. Then, we can select only those ports that have forwarding entries for other network devices.

If no spanning tree exists for the topology of the set of switches, then the network is not functional because of broadcast storms [27]. Therefore, we can assume that each forwarding entry follows a branch of a spanning tree. Each port in p knows the MAC addresses for all of the network devices connected to it, up to the next switch(s) in the hierarchy. For the network shown in Figure 1, here is the port-forwarding table:

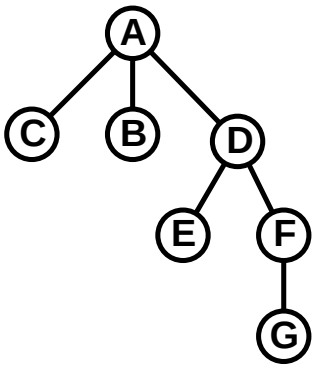

Figure 1: Simple switched network.

A = ((C), (B), (D,E, F,G))

B = (A,C,D,E, F,G))

C = (A,B,D,E, F,G)

D = ((A,B,C), (E), (F,G))

E = (A,B,C,D,F,G)

F = ((A,B,C,D,E), (G))

G = (A,B,C,D,E, F)

This data is also shown in a visual form in Figure 2. An algorithm can then be written to reconstruct the topology using the MAC forwarding table. Consider each node N to be a tree with one element. The algorithm is as follows:

1. Consider each N in T
2. Consider each port p_n in N
3. Consider each M in T where M != N
4. If the tree M contains all of the nodes in the direction of p_n, then remove M from T, and add M as a child of the tree N
5. Repeat until no more changes are made to T, or T only contains one tree.

Thus, the example given in Figure 1 can be solved in three passes. Figure 2 shows the initial setup, where the circular nodes are trees, and the rectangular nodes represent forwarding tables. Figure 3 shows the first pass, which shows the leaf nodes being absorbed into the other trees. Figure 4 shows the second pass, which moves node A into the tree at node D. The last phase, in Figure 5, shows node F moved into the tree with a root of D, and the algorithm terminates, because you have a single tree. While it is not technically the same tree as we started out with, because it is actually a spanning tree of a graph, the graph they represent is indeed the same.

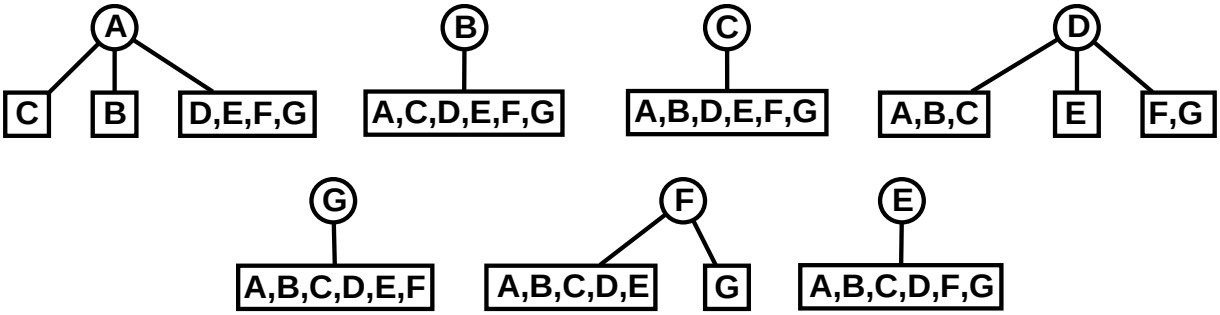

Figure 2: Initial data obtained by polling the MAC address-forwarding database on layer 2 devices.

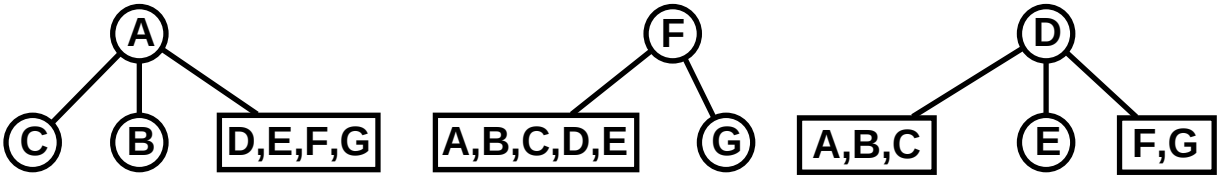

Figure 3: First pass: All the leaf nodes are absorbed into the other nodes.

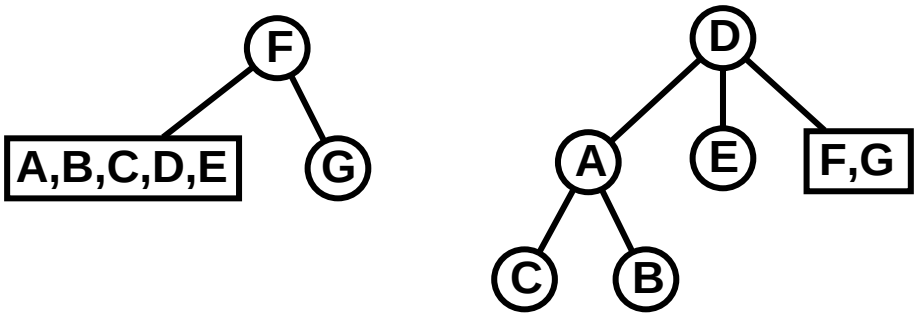

Figure 4: Second pass: Tree A gets placed into tree D.

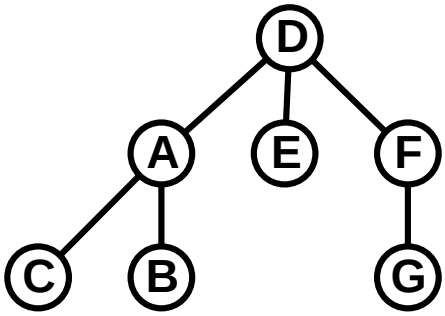

Figure 5: Last pass. Tree F is placed into tree D. Original spanning tree is recovered.

It is possible to infer the existence and position of an unmanaged switch if this algorithm completes, and there are p_n's without children. If we can find a set S of N's in T that compose all of the nodes in the direction of p_n, then we know there is at least 1 unmanaged switch connected to p_n that also connects to the trees in S. If we use the same topology as is shown in Figure 1, except that we assume that we are not polling switch D, then the completed algorithm is shown in Figure 6, which implies the network shown in Figure 7. This algorithm follows the logical topology and not the physical connection of the switches, but if there are no, or simple VLANs, then the two are the same. This is the only method capable of producing a complete tree with missing information. While the complete tree is not necessarily correct, its inconsistencies are predictable, and the information can be used in combination with other methods to produce a more accurate tree.

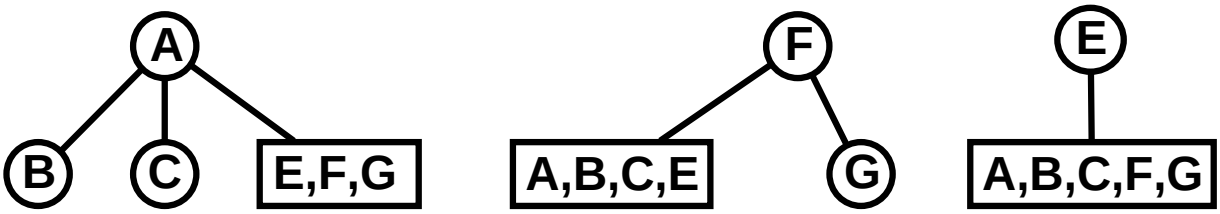

Figure 6: Without the knowledge of node D, this is as far as the algorithm can get. Because the forwarding tables of all three trees contain all the nodes from the other two trees, it is obvious that they are connected via an unknown bridge.

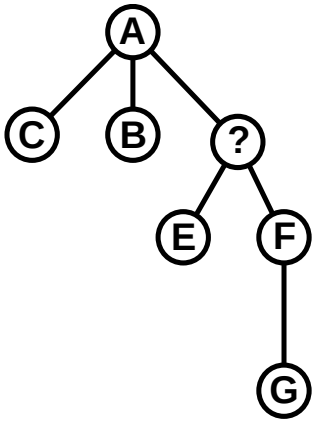

Figure 7: Tree inferred using the MAC forwarding based algorithm.

2.5.4 Combination of these techniques

Each algorithm will produce the same tree on a well-managed network. However, if the network is mismanaged, or has unmanaged devices, they will produce slightly different trees. Consider the network in Figure 8. The MAC forwarding algorithm produces the tree in Figure 7. The presence of the missing node is inferred; however, there is no information on the redundant link. The spanning tree algorithm produces the tree in Figure 9. The tree is produced by using the designated root of each link to infer edges. This means that any known node knows about any link to a higher numbered node. If we merge the two graphs by combining their lists of edges, and trusting the spanning tree method where the edges conflict, the graph in Figure 8 is rebuilt without using any information from the unmanaged bridge.

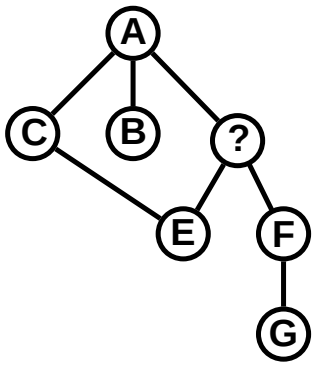

Figure 8: The node marked with a "?" is unmanaged, and the dotted line is a redundant link that has been shut down.

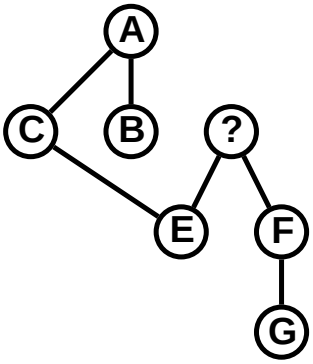

Figure 9: Tree inferred by extracting the spanning tree.

```
done = False
while not done:
     done = True
     for sw in set(nodes.itervalues()):
          if not sw in nodes.values():
               continue
          for port in set(sw.ports()):
               if not port in sw.ports():
                    continue
               if not port in sw.routers:
                    subtree = findsubtree(sw.getrelations(port))
                    if (subtree):
                         sw.childrenlist[port] = subtree
                         if (sw.id in subtree.idtoport):
                            subtree.uplinkport=subtree.idtoport[sw.id]
                         done = False
```

Figure 10: Layer 2 MAC mapping algorithm.

2.5.5 Implementation

This code, figure 10, translates the MAC addresses into the nodes they represent, and sorts them by category. A MAC that represents a router goes into a special case, so that we can attach the resulting network into a layer 3 map. A MAC that exists on its own device is discarded. This happens when somebody plugs a switch into itself, and it confuses the algorithm. The rest are listed as “relations” of the node, since we don’t know what the exact relationship is.

The complexity analysis of this algorithm is done assuming that n is the number of vertices, m is the average degree of its vertices, and o is the average number of vertices per disconnected subgraph. Each pass through the outer loop can remove an arbitrary number of

nodes from the list. The best case (which is unlikely) is that all of the switches are ordered so that every switch analyzed is a child of a switch yet to be analyzed. The average case is that half of the switches analyzed are eliminated on each pass. The worst case is that only one switch is removed each time. Therefore the best case is O(1), the average case is $O(\log_m(n))$, and the worst case is O(n). Looping through the nodes is obviously O(n), and looping through the ports is O(m). The sw.getrelations() call is O(o). All the rest of the operations are either reference assignments, which are O(1), or are sequentially executed with a bigger operation that eclipses them. Thus, the total worst case complexity is O(n * n *m* (o+n)), which reduces to O(n2 *m* (o+n)). The average case is $O(\log_m(n) * n * m * (o + n))$. In real world examples, m and o tend to be very small compared to n. USU's values at the time of writing were n ≈ 708, m ≈ 3, o ≈ 3. o and m tend to stay fairly low, in order to keep the network manageable. Thus the worst case reduces to $O(n^3)$, the average case reduces to $O(n^2 \log(n))$, and the best case reduces to O(n).

3. Grand Unified Lookup Version 3

The interface is very simple, as recommended by Raskin and Tidwell [22][29]. It initially consists only of a simple textbox and a submit button, as shown in Figure 11. It is through this interface that the user can specify a key piece of information for which they need details; i.e., a MAC address, an IP address, or a hostname.

Figure 11: Initial user interface display.

When a user enters a hostname into the top-level query dialog, the user is presented with information presented in Figure 12. The information that is directly known about the entered

hostname is displayed in a hierarchical manner. Most of the information can be shown as a label, which is displayed in bold, and the value or values associated with it. The values are separated using lines in a manner that outlines the relationship between the different pieces of data. For instance, the logical entity girr.usu.edu contains contact information, an IP address, a MAC address, and a host info entry. The contact information also contains a list of entries, which are similarly displayed.

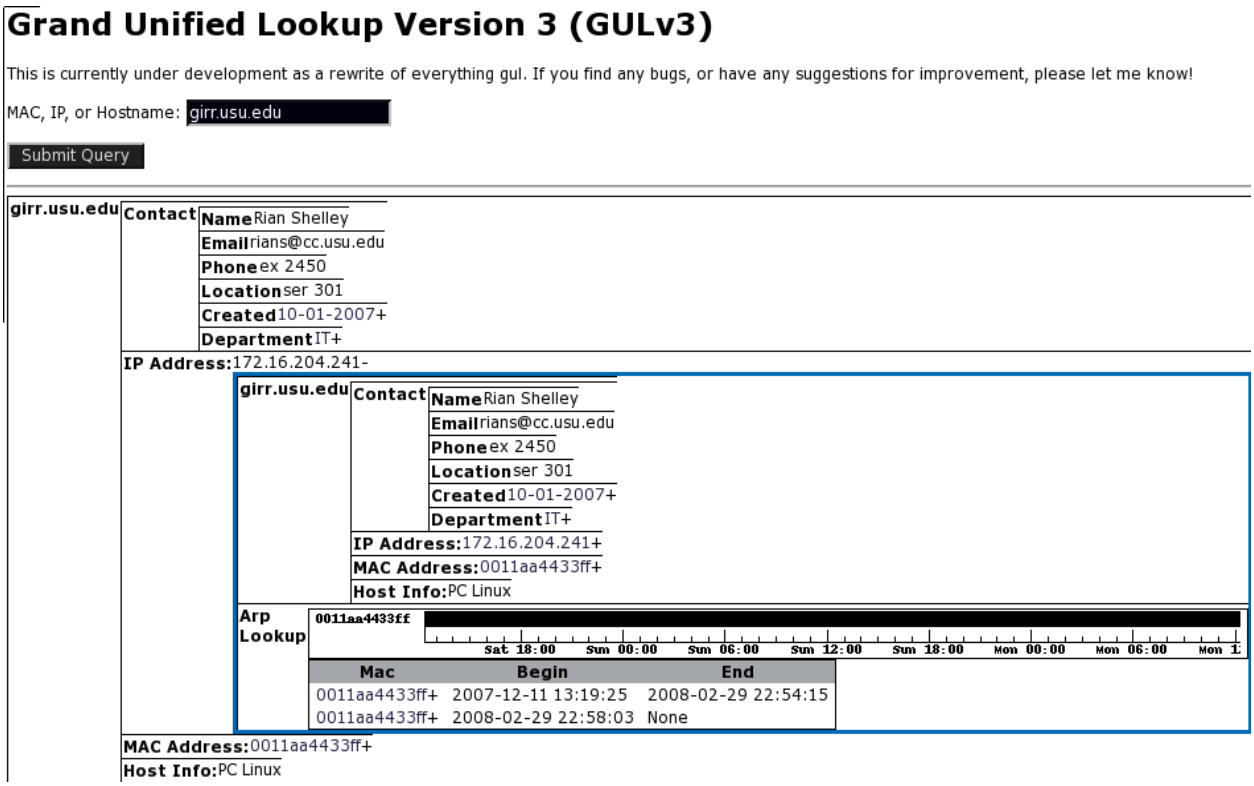

Figure 12: Host-based query displaying all known information related to a specific host. This shows the goal of GULv3 in collating information available from a wide array of disparate sources.

When a user makes a query on a user's last name, the results will be as shown in Figure 13. The search on a user's last name accesses data from the organizations registration database and identifies registrations matching the entered user's name. The hostnames of the matching registrations are hyperlinks allowing immediate searches to be performed on those names to extract information specific for those hosts; i.e., a hostname search is performed as discussed above.

Figure 14 shows two characteristics of the environment. First is the ability of GULv3 to support the entry of MAC addresses into the query dialog. Second is the ability for the query engine to support the entry of partial names and listing all matches. As shown previously, the matches are all provided as hyperlinks such that a more detailed search can be performed on the desired entry.

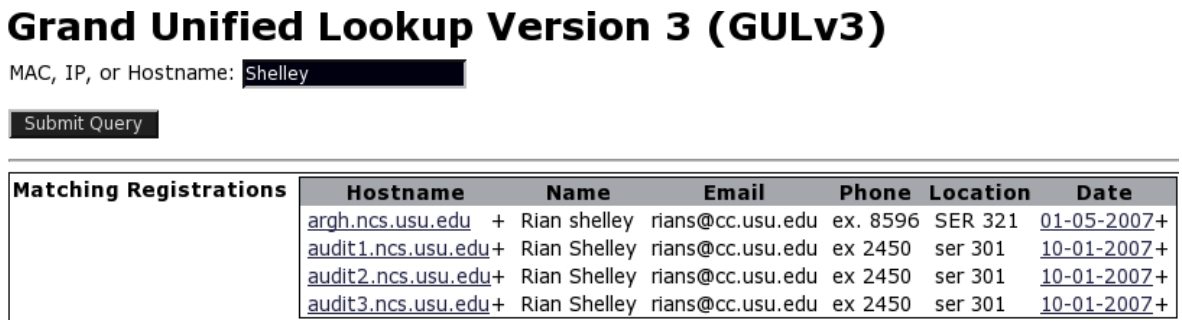

Figure 13: Search in registration database for responsible agent's last name.

From the partial match in figure 14, the user could select one of the entries to do a more complete search. Alternatively, if a full MAC address is known then that could be entered directly. The results of such a MAC address search are shown in figure 15.

Figure 14: Partial match on an entered MAC address. This exemplifies the ability for GULv3 to handle incomplete names in the search engine.

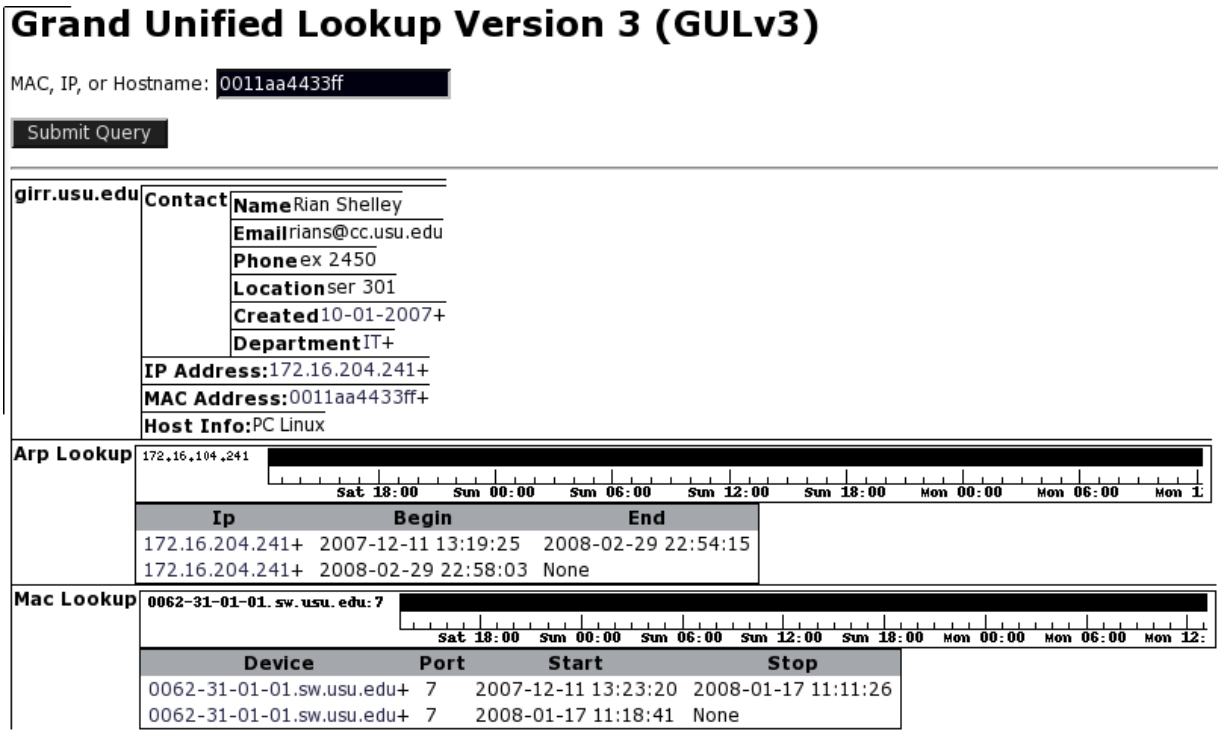

Figure 15: Showing all details of a matched query based on MAC addresses. This could for instance be the next step after clicking on one of the links shown in figure 14 from a partial match.

Figure 16 shows the results of performing a search for a managed switch. Additionally, this example shows the automatic generation of network topology information. Only a partial topology is shown here. Namely, only the topology information directly related to the searched for switch is shown. The difference in available information is also evident when compared with other searches such as that for hostnames or MAC addresses.

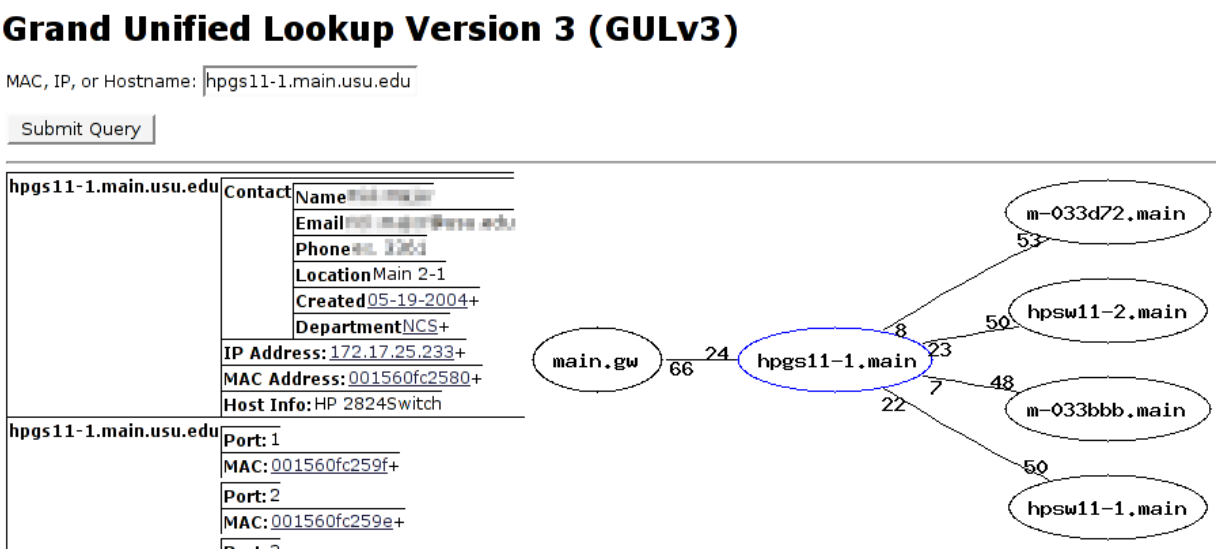

Figure 16: Search for a hostname that turns out to be a managed switch. This example also shows the automatic topology generation of nodes connected directly to this managed switch.

Figure 16 shows a list of ports on a network. The map contains the devices, the connections between them, and the port numbers for the connections. The searched device shows up as a different color than the rest, and serves as the root of the displayed tree. The exception is that the immediate parent of the searched node is also shown. Any of the nodes on the map may be clicked on, which will run a search on it. In this way, we can traverse the topology of the network entirely by clicking on the nodes on the map. The current implementation maps routers, switches, and wireless access points. Internally, the wireless access points are treated like switches.

Inspection of figures 12 and 15 show a graph identifying the IP address associated with a MAC address at any point in time. In those two examples, the specified MAC addresses were always associated with the same IP address. Alternatively, figure 17 shows an example in a DHCP network. In this example, the same IP address is being allocated to different MAC addresses over time. The exact time intervals during which the IP address was associated with each of the MAC addresses can quickly be determined.

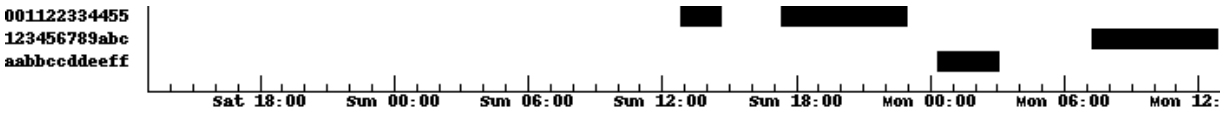

Figure 17: A dynamic IP used by different MAC addresses during different intervals.

A second example of this ability is shown in figure 18. Here a single MAC address appears to be assigned multiple IP addresses simultaneously. This is an example of ARP poisoning and exemplifies the ability of GULv3 to rapidly identify problems on the network simply by doing searches for systems that appear to be having problems.

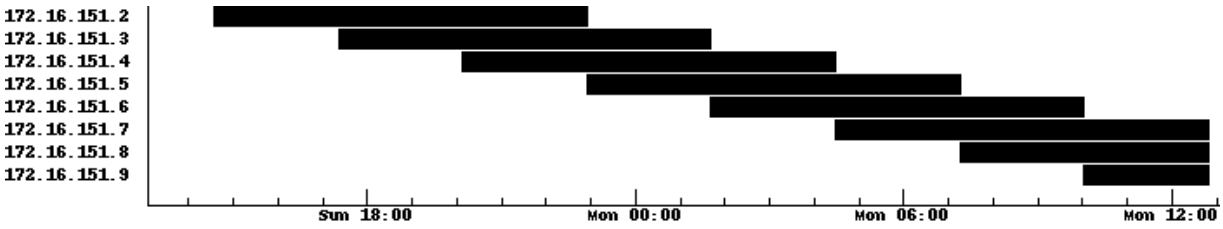

Figure 18: Evidence of ARP poisoning. A single MAC address has multiple IP addresses during overlapping time intervals.

This also exemplifies the applicability of the environment to supporting wireless access points and mobile users in an integrated environment [2]. One of the most challenging aspects of managing such environments is identifying what users were connected to a particular access point at a particular point in time, or identifying what user had a specific IP address at a particular point in time. Analysis of both of these scenarios is supported by GULv3. This management assumes both MAC address registration (validation). MAC addresses can obviously be spoofed, which would be detectable when both the rogue wireless device and valid mobile device connect to the network at the same time. The topology view would allow immediate identification of the location of both devices at a point in time for attempted remediation; though the mobility of such devices can make it difficult to locate the user.

4. Use Cases

The following sections discuss use cases for which GULv3 was been designed to handle. This is in addition to the use-case provided in the introduction to provide background for this work.

4.1 Looking Up Registration Information

The simplest use case is looking up registered information about a computer, and perhaps seeing if the registration varies from what is really happening. Searching for a computer by hostname, MAC address, or IP address returns contact information, and addresses associated with the search term.

4.2 Tracking Rogue Wireless Access Points

Currently, USU has a campus-wide 802.11b/g implementation. The managed wireless access points report the MAC address of any rogue wireless network on campus. Frequently these networks are broadcast using off-the-shelf wireless routers that also do NAT [26][28]. These devices typically have a different MAC addresses on the network-facing side than on the client-facing side. GULv3 can search the first 10 digital of the MAC address reported by the campus AP. This finds any MAC address that matches the first 10 digits, and thus yields both MAC addresses used by the NAT device. Expanding the MAC address plugged directly into a switch returns any information known about that MAC, including what switch port it is plugged into.

4.3 Finding Evidence of ARP Poisoning

A malicious machine on a network may try to poison the ARP cache of the router [31]. Evidence of this shows up in GULv3. A targeted IP address shows up as having more than one MAC address. Over a period of time, a MAC address that is probing for data will show that it is using different IP addresses, as shown in Figure 4.3. Even if the MAC address is spoofed, it will show up in the switch-forwarding table, and so can be traced to the switch port and possibly the wall jack.

4.4 Searching for Bottlenecks

If a network administrator wants to find the source of a bottleneck, the administrator can find the switch attached to one end of this link, and use the mapping feature to follow it to the other end, or to the border of the network. Thus, the administrator knows which links are being used to transfer the data.

4.5 Verifying Network Topology

After new devices are added to the network, an administrator can search for devices in GULv3 and use it to audit the network topology, thus allowing the administrator to verify the shape and make-up of the network.

4.6 Troubleshooting Network Connections

If a customer's connection to the network is not working properly, an administrator can quickly use GULv3 to determine if the switch is aware of the customer and if the customer is using a valid IP address. This allows administrators to answer many of their own questions, rather than relying on the answers the customer, who is often an inexperienced user, is likely to provide.

5. Test results

GULv3, as described, has been implemented at Utah State University. A test phase was used to find bugs and show areas of improvement. In this test, network administrators and service desk personnel with access to GULv3 were given the opportunity to use the system and provide feedback for its improvement. The users are Utah State University employees who have signed a privacy agreement as part of their positions. A subjective test was used because there has not been a similar tool available prior to the initial inception of GUL. A mechanism has been built into the webpage that will allow users to easily and anonymously submit suggestions for improvement and comments. Many suggestions were implemented as part of GULv3.

5.1 Improvements

A number of suggestions were made for small improvements to GULv3. There was also some positive feedback. The following list of changes to the environment was made specifically at the request of the users.

1. The displayed order of ARP and MAC entries was modified so that the most recent entries are displayed first.
2. The ability to search for partial IP and MAC addresses was included.
3. It was noted where it appeared that old data was being preferred over newer data. It turned out that many searches assumed that there would be only one result, and so used the result of the first row. The database displayed records in the order that they were created, which meant that the oldest record was always on top. The search was modified to do a reverse sort on the timestamp field. Many other searches appeared to be vulnerable to a similar bug, so they were modified either to do the sort, or to only consider records newer than a default interval of 24 hours.
4. It was determined that it was possible to do an SQL injection. This was corrected by using a sanitized version of the member variable of the SearchTerm, which is used to classify and sanitize search terms.
5. The search for registrations by contact information was also included. This feature was done by modifying the PartialSearch class.

As GULv3 remains deployed, we continue to accept comments from the users and improve the environment based on their feedback and needs. The limited number of requested changed/identified issues exemplifies the power of the environment.

5.2 Performance

The test machine used to run the polling services and the database was a hyperthreaded [6] 3GHz Intel Xeon processor, with 4GB of RAM. Figures 19 and 20 show CPU usage over

time for the database and polling services. The processes tend to use the CPU in bursts. The polling itself is network bound, waiting on the remote device to respond, and uses very little CPU. The saving of the polled information has to process the data, and uses the CPU for a short period. For the most part, the CPU usage stays low, although it can spike if a particularly busy switch is polled. The average CPU usage over a period of 479 seconds was 14.91%. As this was run on the University's live network, these results give accurate information for a realistic organization.

Figure 21 shows the average CPU usage of the polling services and the database backend, compared to how many simultaneous polling services are running. Because of the simple code, with only one polling service, the CPU usage stays relatively low. As the number of polling services increases, the CPU usage increases linearly until about five polling services. After this, the sum of the CPU usage hits 100%. Because of the hyperthreading used, an increase over 100% is still possible, but after about seven polling services, it more or less levels out. Figure 22 shows the relationship between the targets that can be processed per second and the CPU usage. The relation is linear until the system gets strained. Figure 23 shows the disk usage compared to the number of polling services. The reads are apparently non-existent, which can be explained by the disk caching done by the operating system. The writes show the data that is committed to the disk after each poll; it appears to level off at about the same time that the CPU usage does.

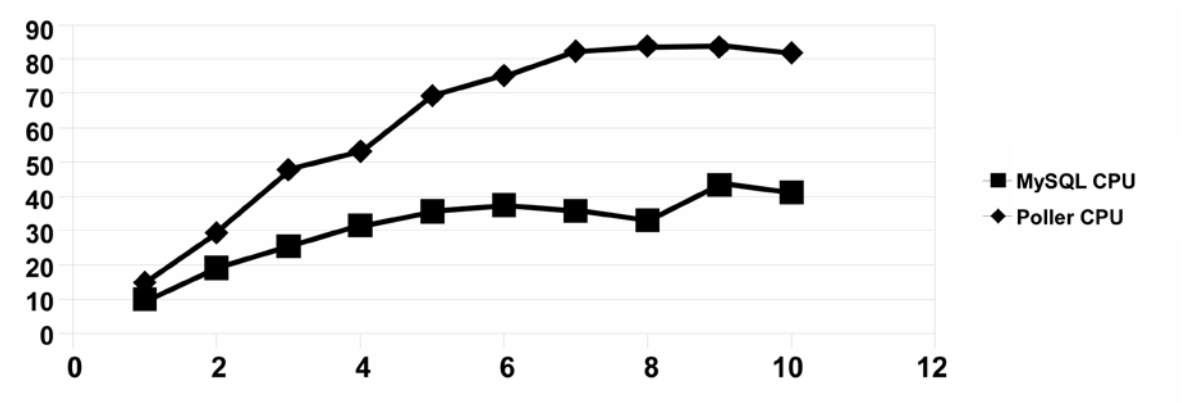

Figure 19: CPU usage for the front end and database processes.

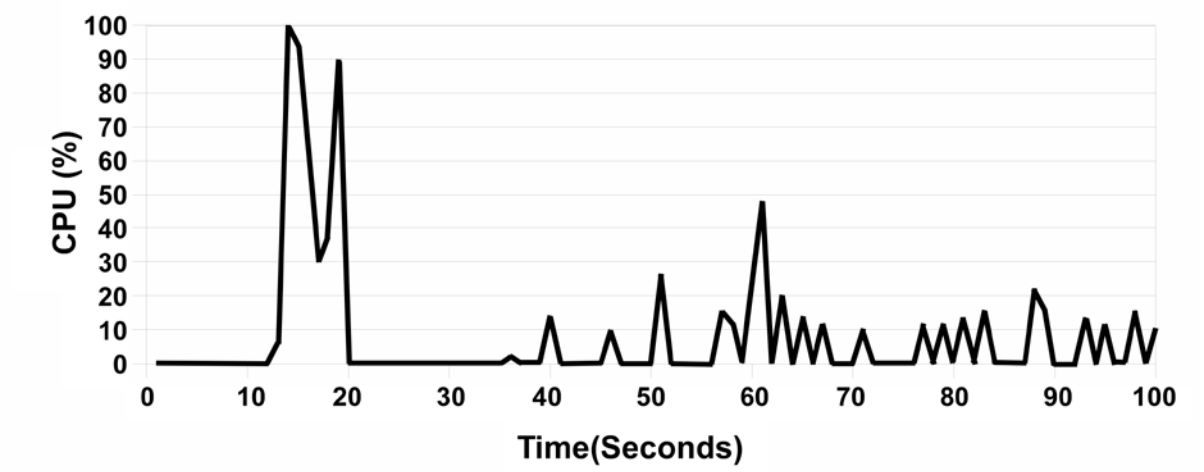

Figure 20: CPU usage for the database process.

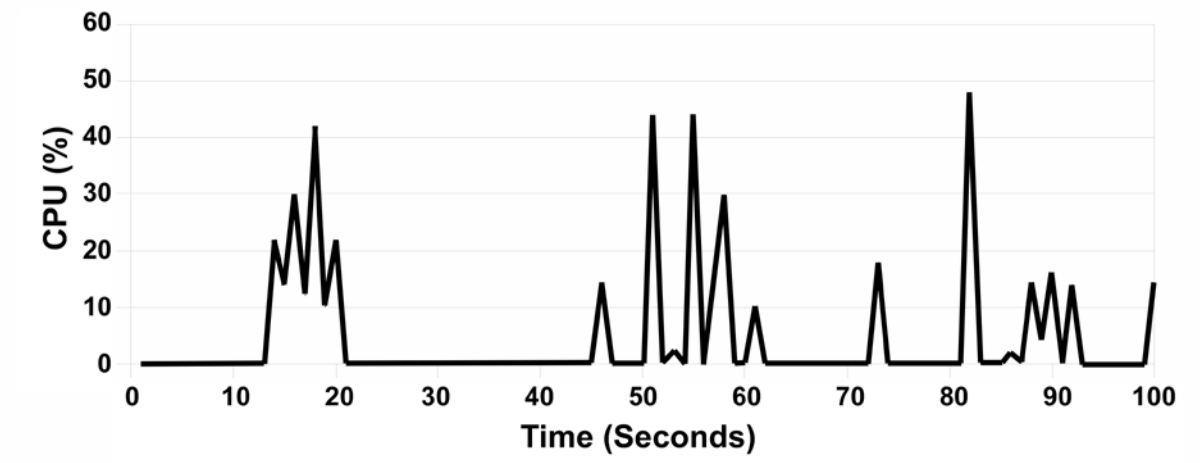

Figure 21: CPU usage for a single polling service.

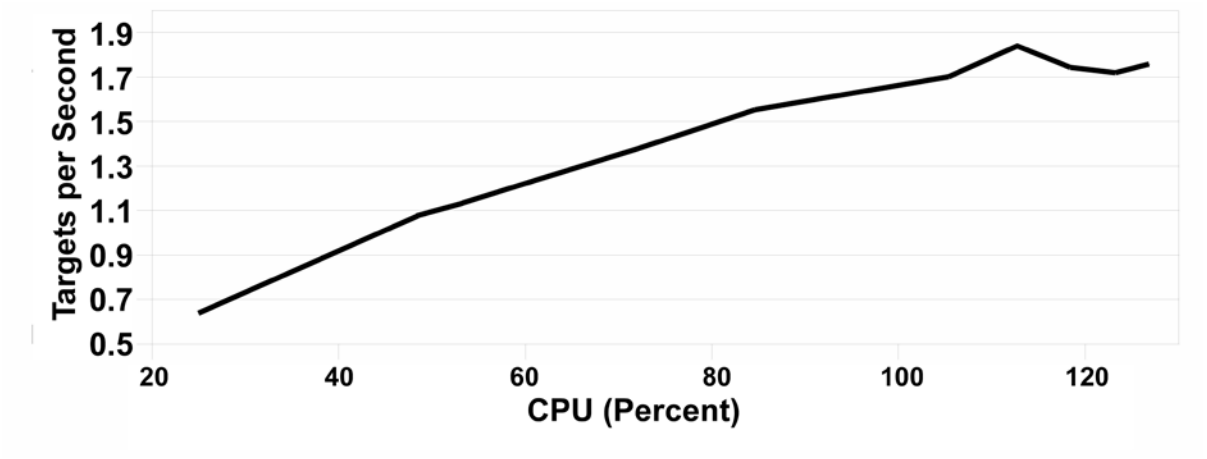

Figure 22: Targets per second related to the CPU processor.

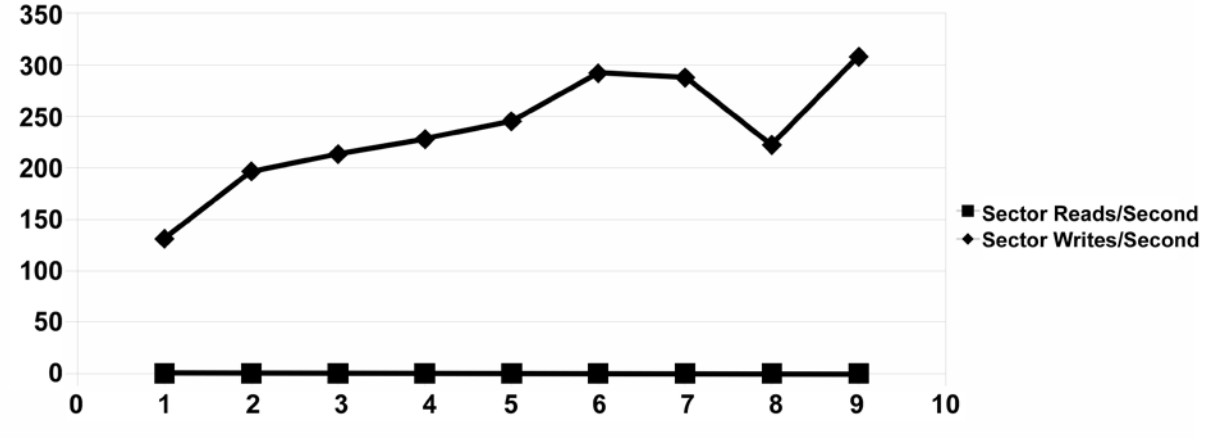

Figure 23: Disk usage compared to the number of polling services.

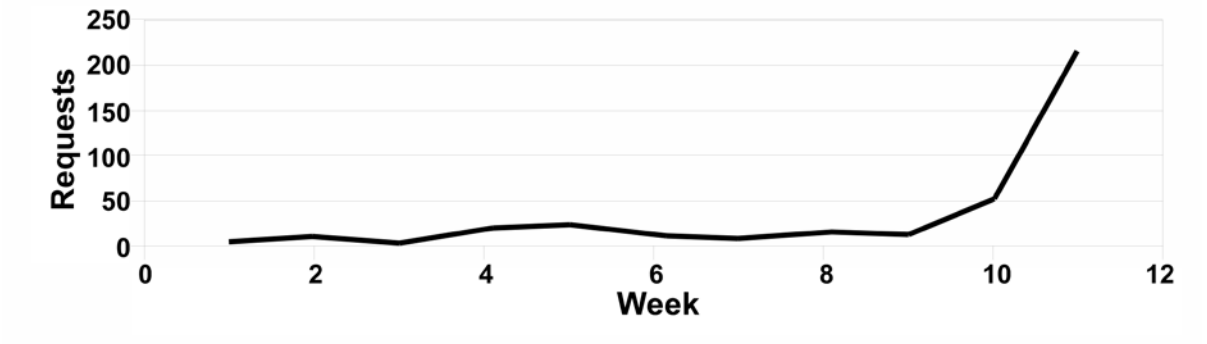

Figure 24: Number of requests per week over the lifetime of GULv3.

5.3 Usage

To get an idea of how useful users considered GULv3, the web page logs were used to determine how many requests per week were processed by GULv3. This is shown in Figure

24. A weeklong period per data point was chosen to minimize the effect due to scheduling differences between employees, and the lack of use that tends to occur on weekends. The IP address where most of the code changes were made was filtered out of the results. Week 10 was the week where the trial period began, and week 11 is where the partial search feature was implemented. The rapid growth in usage shows the two important points. First, the timing of the rapid growth shows the importance of select features in making a tool viable to users. Second, the graph shows the ultimate interest the user base had in the tool and the willingness of the user base to learn how to use the tool and employ it in their activities. This deployability truly shows the value of the tool and its techniques.

6. Future Work

Many of the aspects focused on in GULv3 were selected because of a specific need, or because of their unique solutions. A much more useful tool can be created by including more sources of information and combing them in meaningful ways. As GULv3 continues to be developed, it will be enhanced as it has been, with user feedback driving feature development and bug fixes.

It would be beneficial to include support for more types of network devices that support SNMP, such as wireless access points, printers, servers, etc. It could also be useful to use vendor specific MIBs for devices.

Much of the topology code included in GULv3 is used to discover information that is readily available via LLDP. LLDP would also reveal more information about unknown nodes, and could provide a means of automatically discovering network devices, rather than using a network manager provided list. However, the most benefit would be derived, as described previously, by combining the two methods together to cover up their different shortcomings. Many minor changes to improve the capabilities of the environment are possible:

- In a couple of places, GULv3 currently uses deprecated MIB values. For instance, in The ArpTable class, the ipNetToMediaTable only returns IPv4 addresses, and therefore is deprecated [23]. The ipNetToPhysicalTable should be used instead.
- GULv3 only supports IPv4 addresses but could be easily updated to support IPv6 addresses.
- It would be simple to save the last successful query time of a device, and then indicate the status of that device on the network maps.
- The network maps should also indicate which devices are leaf nodes.
- The interface associated with an ARP entry should be saved. Otherwise, the location of that MAC address may only be known to the vlan, and not to the port itself.
- Unknown MAC addresses on uplink ports are currently discarded. This is to prevent broken devices from littering the database with bogus data.
- Currently, whether or not something is considered a searchable term in the display is governed by a set of regular expressions. It would be more useful if they only showed up as searchable term if it really was in the database.
- There may also be other useful sources of information that could be automatically included in the search. For instance, at Utah State University there is a database maintained that keeps track of which wall jack a switch port is plugged into.

It is possible for a user to set up an unmanaged bridge between two managed ones, and if the user plugs into it, the user's physical location is invisible to GULv3. However, if the information is available from other switches, the MAC address could be saved if it appears on no other leaf port. The network manager would have to follow the topology to determine which port the system is really on. It would be better if the registration information were

obtained more directly than by polling. However, polling is done because of the nature of information available during the time of implementation.

7. Conclusion

This work has provided new capability to aid network administrators in handling many typical network administrator tasks. GULv3's novel user interface capabilities facilitate access to and correlation of the data needed by network administrators. Such capabilities include showing the mapping of IP addresses to MAC addresses over time, generating and displaying the current network topology, hostname search, IP address search, MAC address search, registrant name search, multiple database correlation and display etc. These capabilities assist the network administrator, as seen in the use cases, in such capabilities as locating compromised systems, locating work systems being used inappropriately, identifying where users access the network from, identifying excessive usage, handle failures, identifying inappropriate network reconfigurations, looking up registrant information, etc. More specifically, the complete querying capabilities provide access to registration databases, network device queries, and other previously existing data. The user interface facilitates access to additional information on the query results through hyper-links to provide a complete picture and maintain relationships between all of the information; i.e., all other queryable data elements are provided as hyperlinks.

In addition to the novel user interface, we designed and implemented a novel algorithm for deriving network topologies of layer 2 and layer 3 network devices based on MAC address forwarding tables. This technique is not dependent on spanning trees or any proprietary network discovery protocols. In particular, the MAC forwarding table based mapping algorithm mentioned is a new way of generating a layer 2 topology map without relying on LLDP [11]. Other non-proprietary techniques, such as using time to live (TTL) flags can

create too much noise, be slow on large networks, and are often blocked by typical network firewall rules. We also provided complexity analysis for this developed mapping algorithm.

As indicated, we put substantial effort into acquiring feedback from the target user base. As the functionality increased, especially the inclusion of full query capability, usage increased dramatically. This exemplifies the fact that the target users found the tool useful and greatly simplifies their troubleshooting tasks. Thus, the tool continues to remain deployed and in use. Even with such an increase in usage the performance implications of the tool will not significantly degrade other applications or services. This is a consequence of the fact that while the tool can cause significant spikes in CPU usage, these spikes are very short.

8. Acknowledgements

This work originally performed at Utah State University [24].

9. References

[1] J. R. Allen, “Driving by the rear-view mirror: managing a network with cricket,” in *Proceedings of the 1st conference on Conference on Network Administration*, Santa Clara, CA, 1999, p.1-1.

[2] V. Bahl, R. Chandra, D. Maltz, P. Patel, J. Padhye, and L. Ravindranath, “Towards Unified Management of Networked Services in Wired and Wireless Networks,” Tech Report #MSR-TR-2008-148, MS Research, October 2008.

[3] A. Bierman and K. Jones, “Physical Topology MIB,” in *RFC 2922* (Informational), Network Working Group, Sept. 2000. http://www.ietf.org/rfc/rfc2922.txt.

[4] A. Binczewski, M. Stroinski, and R. Szuman, “Web-based approach to the network physical topology management,” in *IEEE Workshop on IP Operations and Management*, Dallas, TX, 2002, pp. 103–107.

[5] S. Branigan, H. Burch, B. Cheswick, and F. Wojcik, “What Can You Do with Traceroute?,” *IEEE Internet Computing*, vol. 5, no. 5, pp. 96, Sept./Oct. 2001.

[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin, “Simple Network Management Protocol (SNMP),” *RFC 1157* (Historic), May 1990. http://www.ietf.org/rfc/rfc1157.txt.

[7] L. Chao (Editor), “Special Issue on Hyper-threading technology,” *Intel Technology Journal*, vol. 6, no. 1, Feb. 2002.

[8] Cisco, *Cisco discovery Protocol*, 2002. http://www.cisco.com/univercd/cc/td/doc/product/lan/trsrb/frames.htm#xtocid12

[9] B. Donnet and T. Friedman, “Internet Topology Discovery: a Survey,” *IEEE Communications Surveys and Tutorials*, vol. 9, no. 4, pp. 2-15, Dec. 2007.

[10] ICANN, *Internet Assigned Numbers Authority, ianaiftype-MIB*, Sept. 2007. http://www.iana.org/assignments/ianaiftype-mib.

[11] V. Jacobson, *TraceRoute*, Lawrence Berkeley Laboratory (LBL), Feb. 1989. ftp:/ftp.ee.lbl.gov/traceroute.tar.gz.

[12] T. Jeffree, (WG Chair), “IEEE Standard for Local and metropolitan area networks Station and Media Access Control Connectivity Discovery,” *IEEE Std 802.1AB-2005*, pp.0_1-158, 2005. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1490127&isnumber=32029

[13] D. Levi and D. Harrington, “Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering, and Virtual LAN Extensions,” *RFC 4363* (Proposed Standard), Jan. 2006. http://www.ietf.org/rfc/rfc4363.txt.

[14] W. P. Lidinsky (WG Chair), “IEEE standard for local and metropolitan area networks: overview and architecture. Amendment 2: registration of object identifiers,” *IEEE Std*

802-2001 (Revision of IEEE Std 802-1990), 2002. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=984782&isnumber=21227

[15] Y.-X. Lim, T. S. Yer, J. Levine, and H. L. Owen, "Wireless intrusion detection and response," *Information Assurance Workshop*, West Point, NY, 2003, pp. 68-75.

[16] K. McCloghrie and F. Kastenholz, "The Interfaces Group MIB," *RFC 2863* (Draft Standard), June 2000. http://www.ietf.org/rfc/rfc2863.txt

[17] K. McCloghrie and M. Rose "Management Information Base for Network Management of TCP/IP-based internets:MIB-II," *RFC 1213* (Standard), Mar. 1991. Updated by RFCs 2011, 2012, 2013.

[18] J. Moy, "OSPF Version 2," *RFC 2328* (Standard), Apr. 1998. http://www.ietf.org/rfc/rfc2328.txt.

[19] T. Oetiker, "MRTG - The Multi Router Traffic Grapher," in *Proceedings of USENIX LISA*, Boston, MA, 1998, pp. 141-148.

[20] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," *RFC 1142* (Informational), Feb. 1990. http://www.ietf.org/rfc/rfc1142.txt.

[21] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," *RFC 3176* (Informational), Sept. 2001. http://www.ietf.org/rfc/rfc3176.txt.

[22] R. Presuhn, "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)," RFC 3418 (Standard), Dec. 2002. http://www.ietf.org/rfc/rfc3418.txt.

[23] J. Raskin, *The Humane Interface*, Addison-Wesley Professional, Apr. 2000.

[24] S. Routhier, “Management Information Base for the Internet Protocol (IP),” *RFC 4293* (Proposed Standard), Apr. 2006. http://www.ietf.org/rfc/rfc4293.txt.

[25] R. Shelley, “A Novel Technique of Network Auditability with Managers In The Loop,” Master’s Thesis, Department of Computer Science, Utah State University, Logan, Utah, 2008.

[26] M. Son, Y. Lee, C. Pyo, B. Kim, and J. Lee, “Physical topology discovery in large Ethernet networks,” In *Proceedings of the 9th WSEAS international Conference on Communications*, Athens, Greece, 2005, pp. 1-6.

[27] P. Srisuresh and K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” *RFC 3022* (Informational), Jan. 2001. http://www.ietf.org/rfc/rfc3022.txt.

[28] M. Takefman (WG Chair), “IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Bridges Amendment 5: Bridging of IEEE 802.16,” *802.16k-2007 (Amendment to IEEE Std 802.1D-2004)*, pp.1-14, 2007. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4293121&isnumber=4293120

[29] S. Tanenbaum, *The Network Layer in the Internet*, Prentice Hall PTR, 2003.

[30] J. Tidwell, *Designing Interfaces*, O’Reilly Media, Nov. 2005.

[31] N. M. Uithol and V. van Kooten, “Section 2: Network monitoring based on flow measurement techniques,” *SURFnet Research on Networking (RON) project*, Bachelor’s Report. University of Twente, Twente, The Netherlands, Accessed March 2009.

[32] R. Wagner and J. Bryner, “Address resolution protocol spoofing and man-in-the-middle attacks,” *Information Security Reading Room*, 2006. http://www.sans.org/reading room/whitepapers/threats/474.php.

Dr. Erbacher is an Assistant Professor in the Department of Computer Science at Utah State University. He was an Associate Editor for the Journal of Electronic Imaging for 7 years, chaired the SPIE Conference on Visualization and Data Analysis for 13 years, is on numerous program committees, and performs extensive reviewing for conferences and journals. His research interests include Digital Forensics, Computer Security, Intrusion Detection, Information and Scientific Visualization, and Computer Graphics. Dr. Erbacher has over 50 publications. Dr. Erbacher spent the summers of 2004 through 2006 at AFRL's Rome Labs developing visualization for intrusion detection techniques for the air force under their summer faculty fellowship program. He received his BS in Computer Science from The University of Lowell in 1991 and his MS and ScD degrees in Computer Science from the University of Massachusetts-Lowell in 1993 and 1998, respectively.

Mr. Shelley is a network specialist working with the networks group of the information technology office. Mr. Shelley received both his BS and MS in computer science from Utah State University in 2006 and 2008 respectively. His expertise lies in network security, network management, and user interfaces for network management.